DELPHI
tanfolyami
jegyzet

Barhács Oktatóközpont
2004.

Delphi elméleti alapok

Bevezetés

Windows alapú alkalmazások fejlesztéséhez jó darabig csak a C nyelv állt rendelkezésre. Mikor a Microsoft kihozta Visual Basic nevű, komponens alapú fejlesztő rendszerét, a Borland is elérkezettnek látta az idejét, hogy - miután már korábban megjelentette népszerű Turbo Pascal programnyelvét Windowsos változatban - vizuális fejlesztői környezettel lássa el a Pascal-t. Így született meg a Delphi, egy integrált fejlesztői környezetben (**I**ntegrated **D**evelopment **E**nvironment) futó valós idejű fejlesztőrendszer. Könnyedén és gyorsan hozhatunk létre segítségével igényes, bonyolult Windows környezetben futó programokat, köszönhetően a RAD (**R**apid **A**pplication **D**evelopment) felületnek. A korszerű fordító technika eredményeként optimalizált és gyors futású programot nyerünk. A Delphi ugyanis a forráskód lefordítása után valódi, végrehajtható gépi kódot állít elő (szemben pl. a Visual Basic interpreter típusú programmal, mely a forráskódot futási időben értelmezi).

Programozás: A Delphi programozásához újszerű, vizuális fejlesztői stílus "illik", a hagyományos programnyelvekben járatos programozóknak új programozási technikához kell hozzászokniuk. Ennek ellenére - minthogy a Delphi a Pascal nyelvre épül - alkalmazásához szükség van bizonyos Pascal ismeretekre, nem véletlen tehát, hogy a korábbi modulok egyikében már megismerkedtünk a Pascal környezetével. Az Object Pascal programozói nyelvezete az angolra épül, s maga a Delphi feliratok, menük, üzenetek stb. is angolul szerepelnek. Ezért előnyt jelenthet az angol nyelv bizonyos szintű ismerete.

Jegyzetünk a kezdő felhasználók számára ad áttekintést a **Delphi 5.0** 32 bites rendszer használatáról. Teljes körű ismertetésről természetesen nem lehet szó, a témáról könyvek tucatjai jelentek meg. Ezért a részletesebb információkat referencia könyvekből kell beszerezni.

Javasolt irodalom:

Paul Kimmel: Delphi 6 fejlesztők kézikönyve (Panem kiadó)

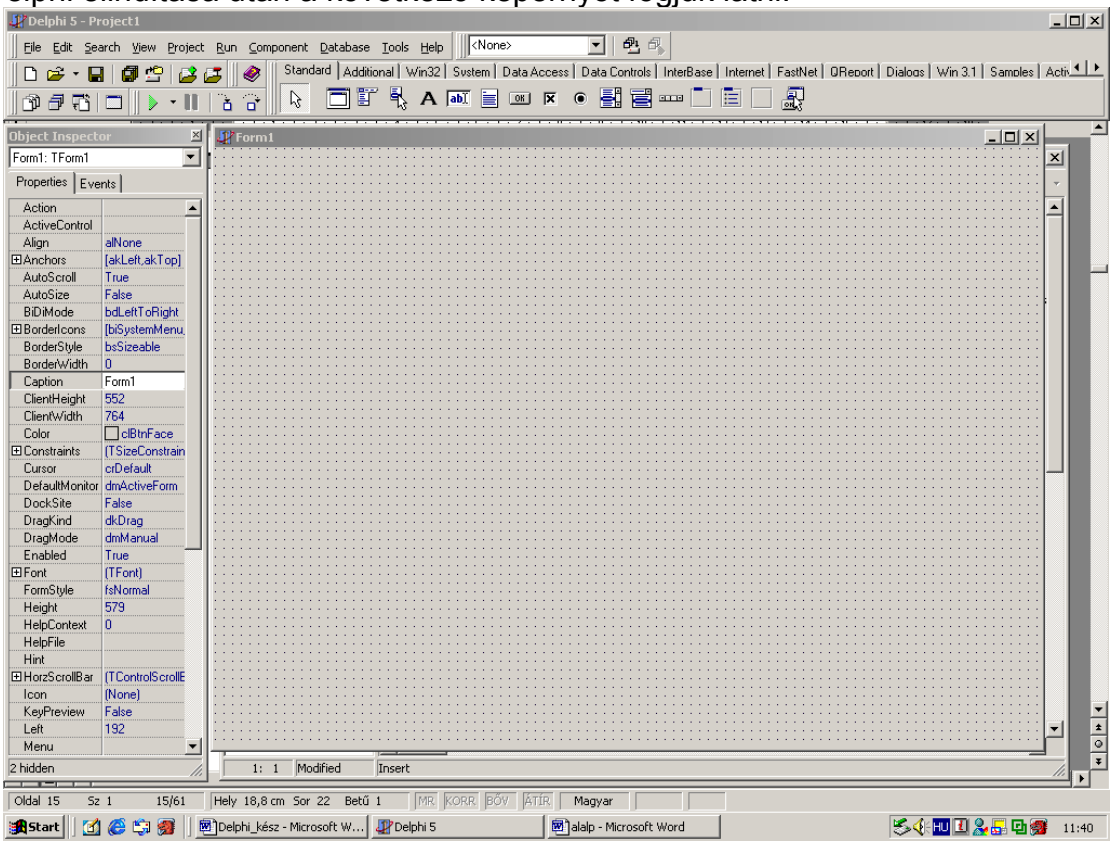
Marco Cantu: Delphi 5 (két kötetes Kiskapu kiadvány)

Telepítés: A Delphi telepítése során rendszergazdai jogosultságokkal kell rendelkeznünk, ugyanis a telepítés során a jogosultságok hiányában előfordulhat, hogy a komponensek nem, vagy nem megfelelően jelennek meg. További problémát jelent majd, hogy a Standard telepítés során a telepítő az Interbase-nek csak a kliens változatát telepíti fel, ami az adatbázis-kezelésnél a későbbiek során problémát jelent. Ezt elkerülendő, a telepítéskor az Interbase felületének teljes telepítését kérjük, hogy gépünk Server-ként is működhessen.

Ha a telepítés sikeresen befejeződött, a startmenüben a programok között megtalálhatjuk a Delphi 5-öt, az Interbase kezelőszerveit, és az InstallShield Express programot, mely a Delphiben készített alkalmazásainkhoz kínál telepítő készletet. Ezen kívül a vezérlőpultban megjelenik még a BDE (Borland Database Engine) adminisztrációs felület is, ahol az adatbázis-kezeléshez szükséges beállítások végezhetőek el.

1. A Delphi megjelenése

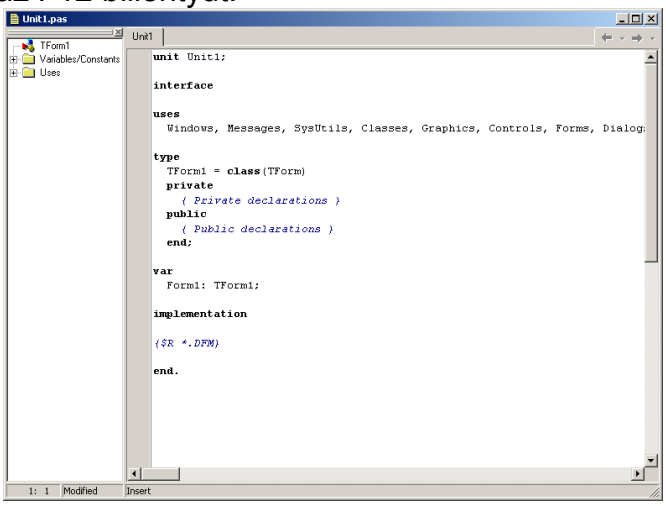
A Delphi elindítása után a következő képernyőt fogjuk látni:



A Delphi fejlesztőkörnyezet négy fő területből áll:

- a **fejrészből**, ami a menüt, az eszköztárakat, és a komponens palettát tartalmazza,
- a bal oldalt látható **objektum-felügyelőből** (Object Inspector),
- a **formszerkesztő** (vagy űrlapszerkesztő) ablakból,
- és az űrlapszerkesztő által részben eltakart **kódszerkesztőből**.

Ez utóbbi úgy válik láthatóvá, ha a formszerkesztő alatt látható részre kattintunk, vagy megnyomjuk az F12 billentyűt.

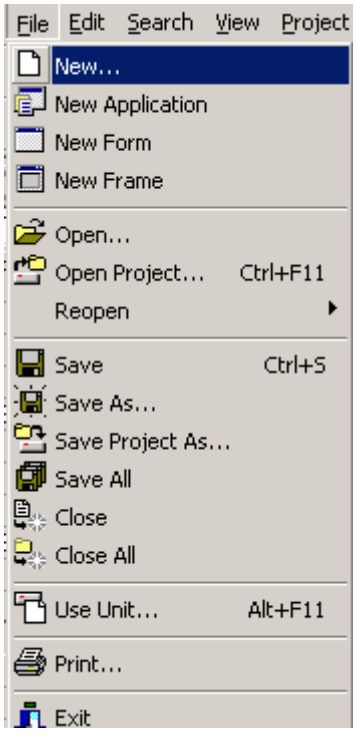


1.1. A Delphi munkaterület fejrésze

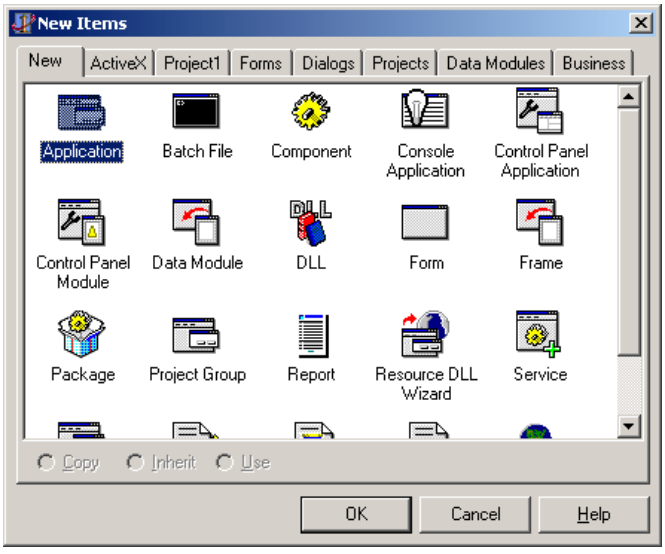
1.1.1. A menü

Bármilyen szároznak is tűnhet a következő rész, a menük többé-kevésbé részletes ismertetése kikerülhetetlen. Előzetesen csak annyit kell megemlíteni, hogy a Delphi menüje is helyzetérzékeny, azaz esetenként a legördülő menülista tételei attól is függnek, hogy éppen milyen műveletet végzünk.

A File menü



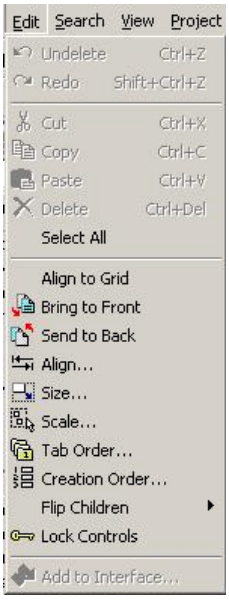
– New...: A menüpont kiválasztásakor egy összefoglaló jellegű, többlapos párbeszéd-ablak jelenik meg (New Items), ahonnan új projektet, vagy más elemeket választhatunk.



A New lapon új projektet nyithatunk meg, vagy pl. űrlapot (formot) vagy forrást (unitot) adhatunk a projekthez. A Project1 lap a már kész formok átalakítását, belőlük új formok előállítását teszi lehetővé. A Dialogs lapról néhány típus párbeszéd-ablak készítéséhez nyújt segítséget., A Projects oldal a több dokumentumból (MDI Multiple Document Interface) és az egy dokumentumból (SDI Single Document Interface) álló felületet tartalmazó projekthez nyújt segítséget. A Data Modules lapon pedig az adatkezelő formok készítéséhez kapunk segítséget.

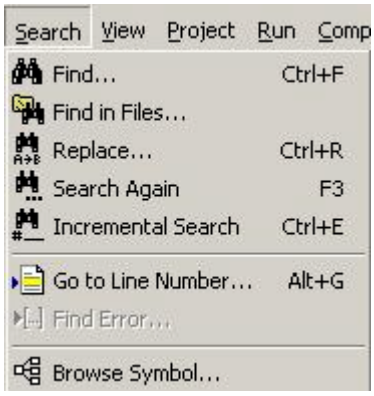
- New Application: Innen egyszerűbben nyithatunk új projektet.
- New Form: Új formot (űrlapot) adhatunk a projecthez.
- New Data Module: Adatbázis jellegű formot kezdhetünk építeni.
- Open...: Létező unitot vagy formot nyithatunk meg, amit egyúttal a projektünkhöz hozzá is adhatjuk.
- Reopen: A legutóbb használt néhány projekt ill. fájl nyitható meg innen.
- Save: Elmenthetjük az aktuális fájlt.
- Save As...: Másik név alatt menthetjük el a fájlt.
- Save Project As...: A projectet új néven menthetjük el.
- Save All: A projecthez tartozó összes fájlt elmenthetjük.
- Close: Bezárhatjuk az aktuális fájlt. Ha még nem volt elmentve, a Delphi rákérdez a mentésre.
- Close All: Az összes megnyitott fájlt bezárhatjuk. A nem mentettekre a Delphi rákérdez.
- Use Unit: A unitok kezelését könnyíti meg (ha több unitból áll a project).
- Print: Nyomtatás.
- Exit: A Delphiből való kilépés egyik lehetősége.

Az Edit menü



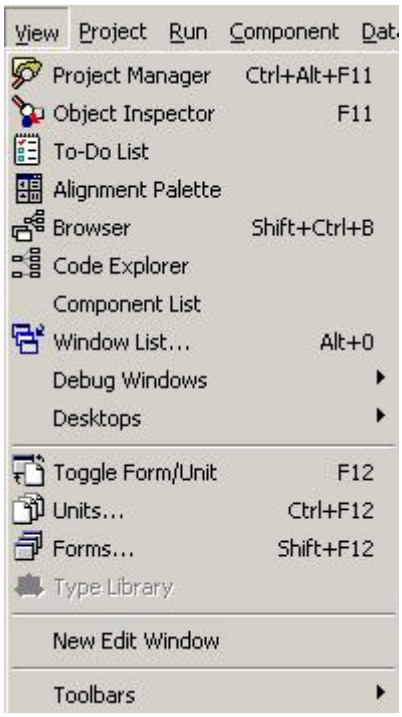
- Undelete vagy Undo és Redo: A szokásos visszavonás-visszaállítás pár.
- Cut: Kivágás.
- Copy: Másolás.
- Paste: Beillesztés.
- Delete: Törlés.
- Select All: A szövegszerkesztő ablak minden sorát kijelöli (formszerkesztőben minden elemet).
- Align to Grid: Rács, amelynek segítségével pontosan helyezhetők el a komponensek a formokon.
- Bring to Front: A kijelölt objektumot a többi fölé helyezi, ...
- Send to Back: ... ez pedig alá.
- Align: Párbeszédablak segíti a precíz komponens-elhelyezést.
- Size: A kijelölt objektum pontos méreteit állíthatjuk be.
- Scale: A komponens nagyíthatóságáról tudunk itt intézkedni.
- Tab Order: Egy formon azt lehet beállítani, hogy a Tab billentyűvel milyen sorrendben lépkedjük végig a komponenseken.
- Creation Order: A nem vizuális komponensek létrehozása.
- Flip Children almenü: a kiválasztott, vagy az összes elem elrendezésének módosítására ad lehetőséget.
- Lock Controls: A form elemeinek rögzítésére való.

A Search menü



- Find...: Szöveget kereshetünk vele. A megnyíló párbeszédablakban megadható a szöveg, megadható, hogy megkülönböztesse-e a kis- és nagybetűket (Case sensitive), vagy hogy csak teljes szavakat kell-e keresni (Whole words only). Megadható a keresés tartománya, azaz hogy a teljes szövegben, vagy csak a kijelölt blokkban kell-e keresni (Global, Selected), a keresés iránya előre, vagy vissza (Forward, Backward) történjen stb.
- Find in Files…: Az állományokon belüli keresés.
- Replace...: Szöveget cserélhetünk, a Text to find sorba a keresendő, a Replace with sorba a csereszöveget kell beírni. A keresési funkciónál megismert beállítási lehetőségek itt is megvannak. A Replace all bekapcsolásával valamennyi előfordulást automatikusan lecseréli.
- Search Again: A keresési, illetve csere művelet ismételhető.
- Incremental Search: Ez is keresés, de begépelés közben a keresett szöveghez ugrik.
- Go to Line Number...: A megadott számú sorra ugrik a kurzor.
- Find Error: A fordítás vagy futtatás utáni hiba helyére ugrik.
- Browse Symbol: Szimbólumot kereshetünk (csak fordítás után használható).

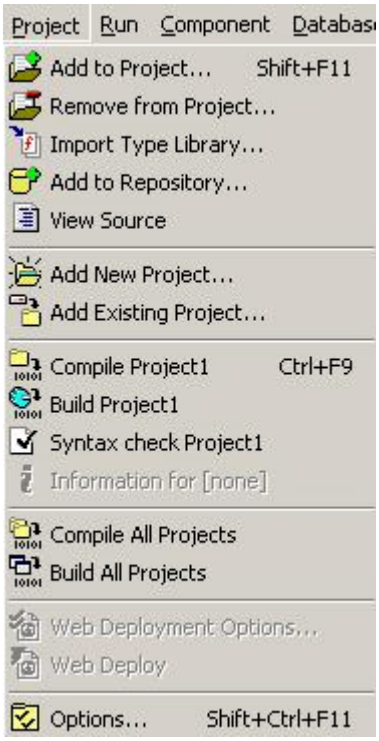
A View menü



- Project Manager: A Project Manager ablak megjelenítésére való. Itt található a project valamennyi állománya az elérési útvonalakkal, nevekkel, bővítési lehetőségekkel.
- Object Inspector: Az objektum-felügyelő ablakot jeleníti meg.
- To-Do List: A tennivalóink listája, emlékeztetők saját részünkre.
- Alignment Palette: A megnyitott ablakban a kijelölt elemeket egyszerűen rendezhetjük el.
- Browser: Csak lefordított program esetén működik. Osztályok, metódusok hivatkozásait, hatáskörét tekinthetjük meg a segítségével.
- Code Explorer: A Unitban (kódunkban) való navigálást teszi egyszerűbbé.
- Component List: Azon komponensek listája, amiket elhelyezhetünk a formon.
- Window List: A Delphiben megnyitott ablakok megjelenítésére való.
- Debug Windows almenü: A beépített Debuggert (hibakeresőt) indíthatjuk el.
- Desktops almenü: A Delphi képes az aktuális Desktop mentésére.
- Toggle Form/Unit: A form és a unit között kapcsolgathatunk.

- Units: A projektben lévő unitok jelennek meg.
- Forms: A projektben lévő formok jelennek meg.
- Type Library: Az objektum típusokról kaphatunk bővebb információt.
- New Edit Window: Új szövegszerkesztő (kódszerkesztő) ablak nyílik meg, mellyel a forráskódú szöveg különböző részeit szerkeszthetjük.
- Toolbars almenü: Az eszköztáraink megjelenítését, testreszabását teszi lehetővé.

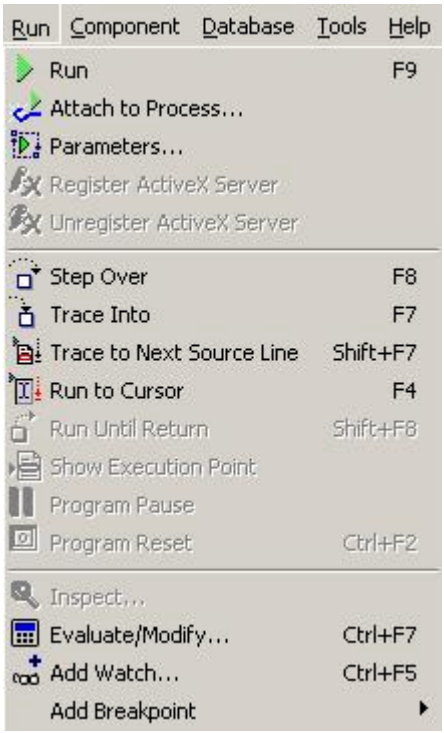
A Project menü



- Add To Project: Formot vagy unitot adhatunk a projectünkhöz, ...
- Remove From Project: ... ezzel pedig eltávolíthatjuk.
- Import Type Library…: Importálhatunk külső objektum típus könyvtárakat.
- Add to Repository...: Ha olyan formot készítettünk, amit máskor is fel akarunk használni, felvehetjük a kelléktárba.
- View Source: A forrás megtekintése.
- Add New Project…: A meglévő project mellé újabbat indíthatunk.
- Add Existing Project…: A létező projectbe meghívhatunk korábbi projectet.
- Compile All Projects: A módosított forráskód lefordítására való.
- Build All Projects: A projekt összes unitját lefordítja.
- Syntax Check: Ellenőrzi a kód szintaktikáját. Futtatás, de fordítás előtt is érdemes elvégezni.
- Information...: A lefordított fájlról szolgáltat adatokat.
- Web Deployment Options…: A Web szerverre készült ActiveX komponensekkel ellátott form konfigurálását teszi lehetővé.
- Web Deploy: A fenti konfiguráció után használható az objektumaink beállításához.
- Options...: Többoldalas párbeszéd-ablakban lehet különböző beállításokat végrehajtani az általunk készítendő projecten.

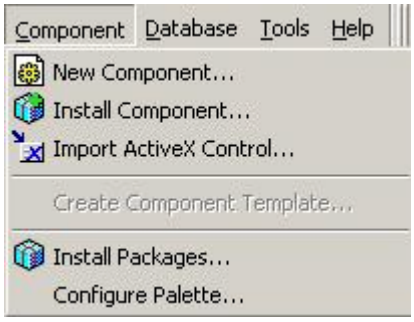
A Run menü

Ebben a menüpontban a futással, futtatással kapcsolatos lehetőségeket találhatjuk meg.



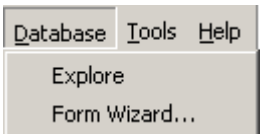
- Run: Futtatja a programot. Ha a program szövege megváltozott, akkor előbb automatikus fordításra és szerkesztésre kerül sor (jegyezzük meg az F9 billentyűt, a futtatáshoz).
- Attach To Process…: Az aktuálisan futó folyamat ellenőrzése.
- Parameters…: Futási paraméterek beállítására való.
- Register ActiveX Server: Az ActiveX szerver regisztrálása,
- Unregister ActiveX Server: illetve annak törlése.
- Step Over: A programfutás követésére való. Szubrutinokba nem megy be (F8).
- Trace Into: A programfutás soronkénti követésére való. Szubrutinok hívásakor a szubrutinok utasításain is végigmegy a program (F7 funkcióbillentyű).
- Trace to Next Source Line: Lépés a következő programsorra.
- Run to Cursor: Nem fut végig a program, csak az aktuális kurzorpozícióig (F4).
- Run Until Return: Az aktív folyamat futtatása a visszatérési értékig.
- Show Execution Point: Megmutatja a következő programsort.
- Program Pause: Felfüggesztés.
- Program Reset: Befejezi a lépésenkénti futtatást.
- Inspect…: A kiválasztott elemről futási időben megjeleníti az információkat.
- Evaluate/Modify...: A programfutás során keletkezett változó értékekre lehet rákérdezni az Expression (kifejezés) mezőben, az értékek a Result (visszatérési érték) mezőben jelennek meg.
- Add Watch...: Egy figyelendő változót lehet beírni, a programfutás során figyelemmel kísérhető az érték alakulása.
- Add Breakpoint...: Töréspontokat lehet beszúrni a programba.

A Component menü



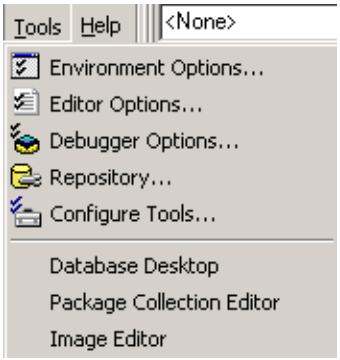
- New...: Új komponenseket készíthetünk ha nem tartjuk elegendőnek a felkínált lehetőségeket.
- Install Component...: Kész, vásárolt, letöltött komponenseket telepíthetünk a Delphi komponenskönyvtárába.
- Import ActiveX Controll…: ActiveX elemek importálása.
- Create Component Template…: Átmeneti komponenskönyvtárak létrehozása.
- Install Packages…: Speciális dinamikus könyvtárak csatolásának lehetősége.
- Configure Palette...: A komponenspaletta testreszabása.

A Database menü



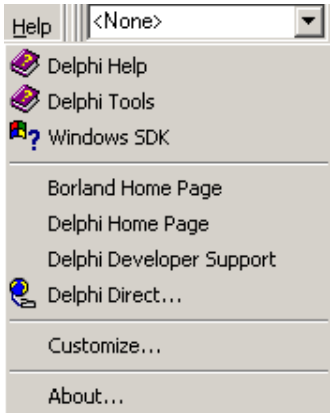
- Explore: Betölti a Database Explorert, amely lehetővé teszi az adatbázis-struktúrákban való keresést, szerkesztést.
- Form Wizard...: Adatbázis beviteli képernyők, formok készítését segíti.

A Tools menü



- Environment Options...: Többlapos párbeszéd-ablakban a Delphi különböző beállításait lehet elvégezni.
- Editor Options…: A szerkesztővel kapcsolatos beállítások
- Debugger Options…: A hibakeresővel kapcsolatos beállítások.
- Repository...: Az Add to Repository paranccsal létrehozott kelléktárhoz való hozzáférés.
- Configure Tools...: A Delphi-hez adott eszközök elérése.
- Database Desktop: Adatbázis kezelési felület indítása.
- Package Collection Editor: A telepített csomagok szerkesztésének lehetősége.
- Image Editor: a Delphi program képszerkesztő-rajzoló eleme.

A Help menü



- Delphi Help: A Delphi 5 súgója, felhasználói szinten.
- Delphi Tools: A Delphi 5 súgója fejlesztőknek.
- Windows SDK: Fejlesztői kézikönyv a Delphihez.
- Borland Home Page: www.borland.com
- Delphi Home Page: www.borland.com/delphi/index.html
- Delphi Developer Support: http://info.borland.com/devsupport/delphi/
- Delphi Direct…: Egy beépülő modul, mely az interneten keres segítséget.
- Customize…: Nyílt segítség, a gépre telepített valamennyi súgó igénybe vételével.
- About...: A minden programhoz járó Névjegy Delphi változata.

Egyáltalán nem fontos, de érdekes: ha az About ablakot megnyitottuk és az Alt gomb lenyomott állapotában beírjuk: team, akkor megtekinthetjük a Delphi létrehozásában közreműködő csapat névsorát. Ha hasonló módon a developers szót írjuk be, akkor a fejlesztők neveit olvashatjuk.

1.2. Az eszközpaletta

Látható, hogy a Delphi 5.0 ablakai a Windowsban megszokott jellemzőkkel bírnak: címsor, ablakkezelő gombok, vezérlőmenü stb. A Delphi címsora a Delphi 5 Project1 feliratot viseli (ez a felirat azonban a különböző állapotoknak megfelelően - pl. mentés után – értelemszerűen változik). A menü 10 tételt tartalmaz, amelyekből szabványos legördülő menük jelennek meg. Az itt található parancsok gyakrabban használható része a menüsor alatt, baloldalon található eszközpalettán is megtalálható ikonként. Ezek a gombsorok természetesen testreszabhatóak, azaz olyan gombokkal egészíthetők ki, amiket munkánk során gyakran használunk, a kevésbé használatosak pedig eltávolíthatók. A gombok funkciójáról súgócímke ad tájékoztatást, azaz ha valamelyikre ráállunk az egérkurzorral, egy keretben lévő felirat (buborékablak) mondja meg, mire is való. Az alapbeállítású nyomógombokat mutatjuk be a következőkben:

	New	Elindítja az új elemek párbeszédablakot.
	Open	Felkínálja a megnyitás ablakot
	Save	Az aktuális UNIT mentése.
	Save All	A UNIT, a Formok, és a Project mentése.
	Open Project	Meglévő Project megnyitása
	Add File To Project	Állomány hozzáadása az aktuális Projecthez
	Remove file from project	Eltávolítja a kiválasztott elemet a projectből.
	View Form	Megnyitja a form-megjelenítő párbeszéd-ablakot.
	Toggle form/unit	A fókuszt a formról a unitra helyezi át, illetve fordítva.

	New form	Új formot ad a projecthez.
	Run	Futtatja a projectet.
	Pause	Megállítja a project futtatását.
	Trace into	Soronkénti futtás.
	Step over	Ez is, de az eljárásokba nem megy bele.

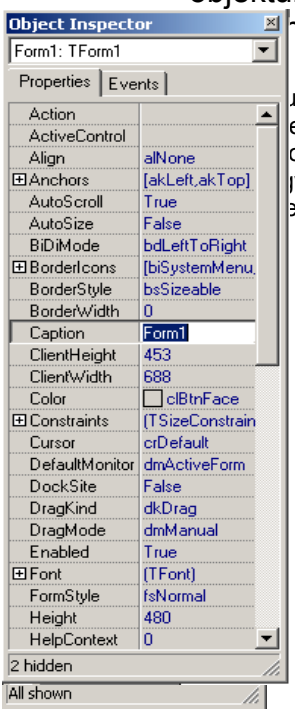
1.3. A komponenspaletta

A komponenspalettán alapesetben 15 lapon, tematikus csoportokba gyűjtve találhatók a Delphi komponensei.

Már a Standard lapon található elemekkel felépíthető egy átlagos form, azaz Windows ablak (menü, gombok, jelölő négyzetek, rádiógombok stb. felhasználásával), az esetleg még szükséges komponensek a további lapokon találhatók. A form szerkesztése úgy történik, hogy a komponenseket az adott gombra kattintva kiválasztjuk, majd a form egy adott helyére kattintva elhelyezzük a form felületén. Az egyes elemek tulajdonságait az Object Inspector-ban (objektum-felügyelő) állíthatjuk be.

1.4. Az objektum-felügyelő

Az Object Inspector voltaképpen egy kétlapos párbeszéd-ablak, ahol beállíthatjuk a form és a komponensek tulajdonságait (név, méret, szín, betűk stb.). Ez történik a Properties nevű lapon. Az Events (események) lapon pedig az egyes elemekhez kapcsolódó eseményekről (pl. egérkattintás, billentyűleütés stb.) intézkedhetünk. Az objektum-felügyelőben mindig a formszerkesztőn aktív (vagyis kijelölt) elem



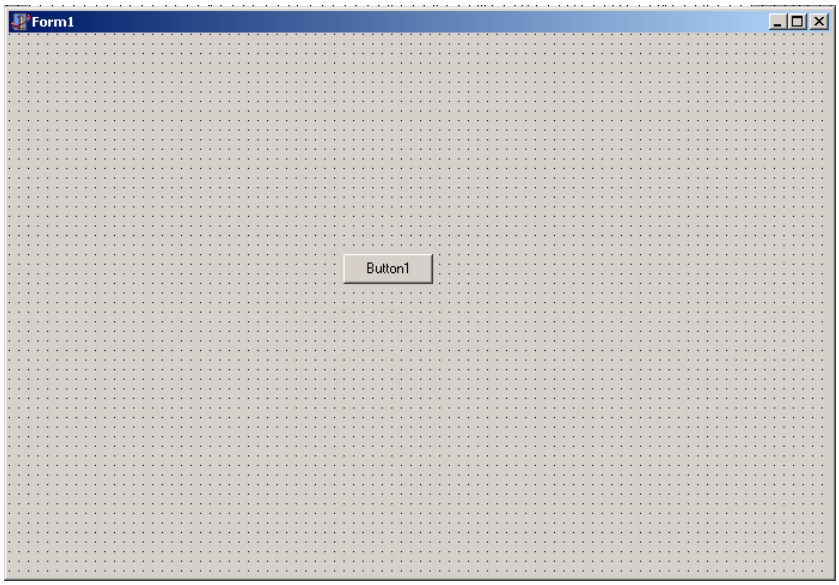
ságai állíthatók be, illetvemódosíthatóak.

ulajdonságok az ABC betűi szerint növekvő endben helyezkednek el, fentről lefelé, így onylag könnyen megtalálhatjuk a változtatni kívánt tulajdonságot feltéve, ha erjük annak angol nevét.

A Delphi objektumainak természetesen eltérő eseményeik lehetnek, így az események részletes felsorolástól itt most eltekintenénk, a példafeladatok során a legjelentősebbeket meg fogjuk ismerni

1.5. A formszerkesztő

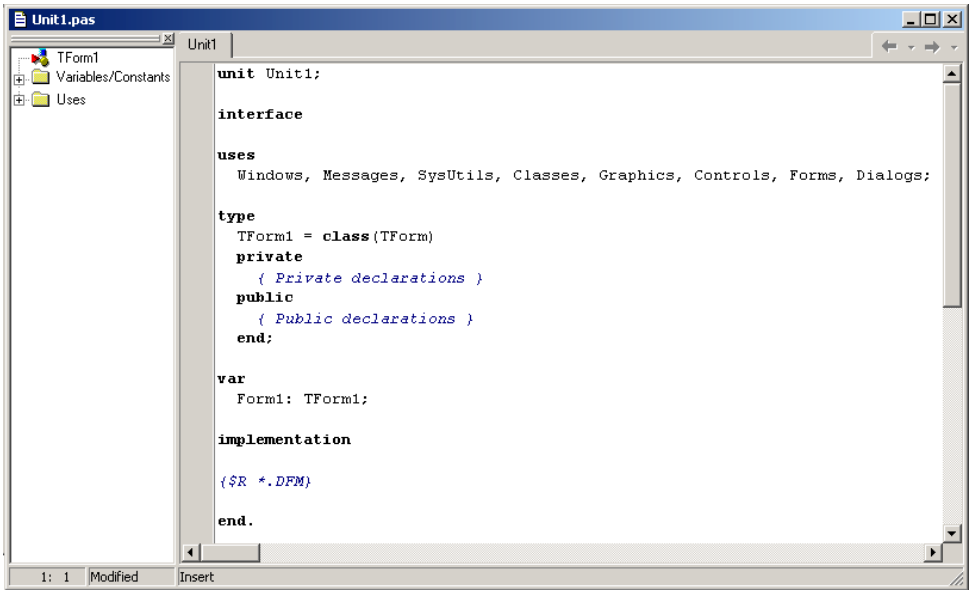
A formszerkesztő az a felület, amelyen a tervezett Windows ablakot kialakítjuk. A formon (űrlapon) kell elhelyezni a menüt, gombokat stb., az objektum-felügyelőben pedig megadhatjuk ezek tulajdonságait. Már maga a form is egy teljes, ámbár üres Windows ablakot takar. Ha lefuttatnánk, egy olyan üres Windows ablakot kapnánk, amit méretezhetünk, áthelyezhetünk, használhatnánk az ablakgombokat, de még a rendszermenüt is.



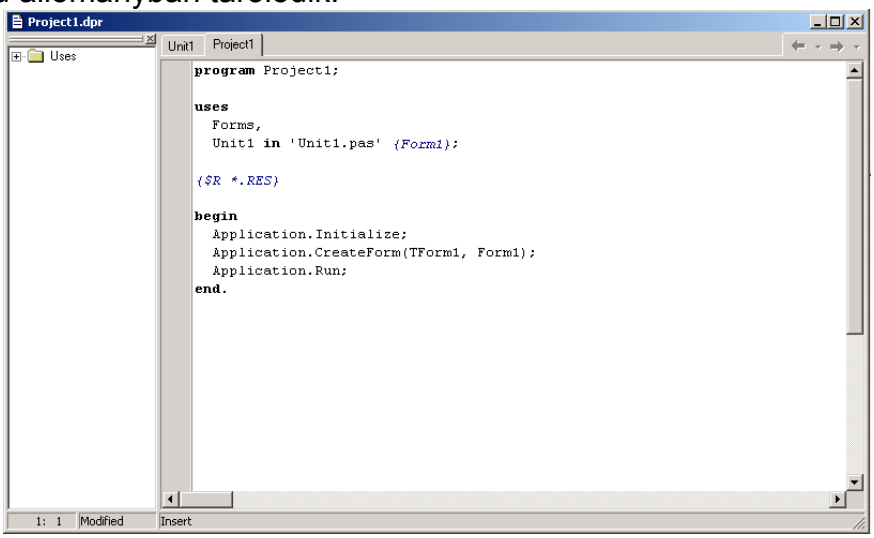
A fenti ábrán egy nyomógomb komponenst helyeztünk az űrlapra. A form felületén a komponensek elhelyezését pontozott rács segíti.
Az ablakmodellek leírását a Delphi .DFM (Delphi Form) kiterjesztésű állományokban tárolja.

1.6. A kódszerkesztő ablak

A kód- (szöveg-) szerkesztő ablak alapesetben a formszerkesztő alatt alig látható. Akkor tűnik elő, ha a kilógó alsó részére kattintunk, vagy megnyomjuk az F12 funkcióbillentyűt. Ennek eredményeként a fent elhelyezkedő űrlapunk előtt megjelenik az addig a háttérben meghúzódó UNIT ablak a következő ábra szerinti formában:



Az ablakban az eddig "megírt" Object Pascal nyelvű kódszöveget láthatjuk. A Delphi ugyanis az általunk vizuális módon szerkesztett form, illetve a komponensek Pascal-kódját rögtön beilleszti az aktuális unit megfelelő helyére. A unitok fájljai .PAS kiterjesztésűek. Minden egyes formhoz tartozik egy unit (de természetesen létezik form nélküli unit is). Egy program alá általában több form és több unit is tartozik. Maga a program (azaz a project) kódja is megtekinthető, ha a View menü UNIT parancsát választjuk, és ott kiválasztjuk a Project1-et. A projekt forráskódja .DPR kiterjesztésű állományban tárolódik.

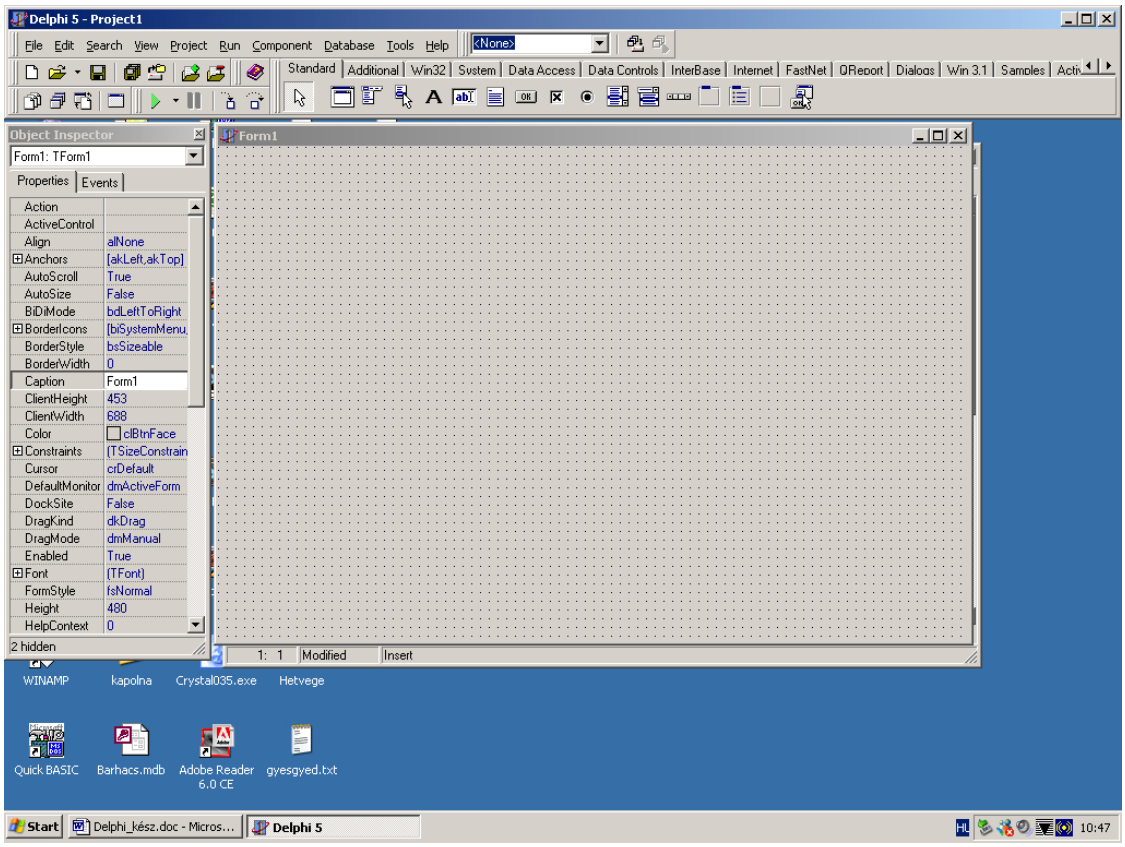


Bár az ábrák szerinti kódszerkesztő ablak a kódszövegnek csak kis részét láttatja (valójában ezek is méretezhető ablakok), annyi máris kitűnik, hogy az Object Pascal szövegszerkesztője - a Turbo (Borland) Pascal 7.0 haladó hagyományait követve - automatikusan kiemeli a programnyelv kulcsszavait, a megjegyzéseket pedig dőlt kék betűkkel írja.

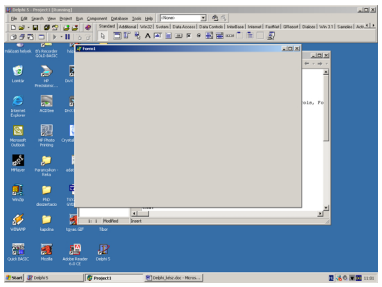
Első Delphi programunk:

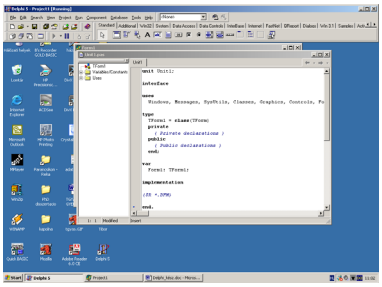
Miután a Delphi kezelésének alapvető információit elsajátítottuk, itt az ideje, hogy elkészítsük első önálló alkalmazásunkat. Ennek keretén belül ismerhetjük meg a Delphi felépítését, a különböző állományok szerkezetét, azok mentésének alapvető szabályait, és a névkonveciókat, vagyis az objektumok elnevezési szabályait.

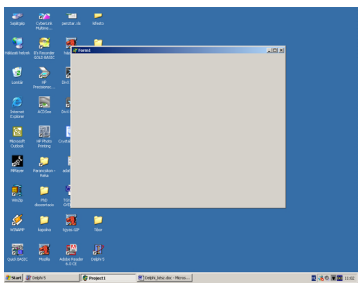
Indítsuk el a Delphit az asztalon található Delphi 5 parancsikon segítségével. Miután betöltődött az összes szükséges rendszerkomponens, a következő képernyőt fogjuk látni:



Tulajdonképpen már el is készült egy önállóan futtatható szabványos Windows ablak, amit meg is tekinthetünk, ha a Run menüpont Run parancsát választjuk (vagy megnyomjuk az eszköztáron található Run ikont, esetleg leütjük az F9 billentyűt). Ennek eredményeképpen elindul az alapértelmezetten Project1-nek nevezett alkalmazás, ami nem más, mint egy teljes értékű Windows ablak, a szokásos tulajdonságokkal (címsor, vezérlőmenü, átméretezhető felülettel).
Ha figyelmesen megvizsgáljuk látható, hogy a megjelenő ablak hátterében a Delphi kezelőfelülete változatlanul megtalálható, de futásidőben az Objektum felügyelő nem látható. Amennyiben a Delphit előtérbe hozzuk, akkor láthatóvá válik a UNIT, amennyiben pedig a tálcára tesszük a Delphit, akkor pedig csak a futó alkalmazás lesz látható:

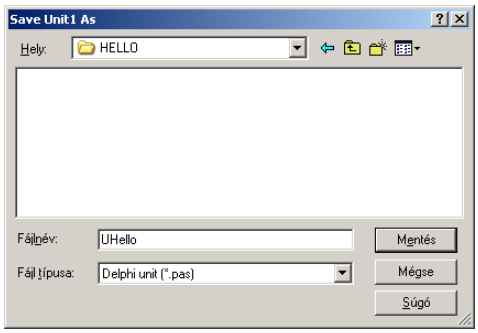

Alapesetben


A Delphit aktiválva

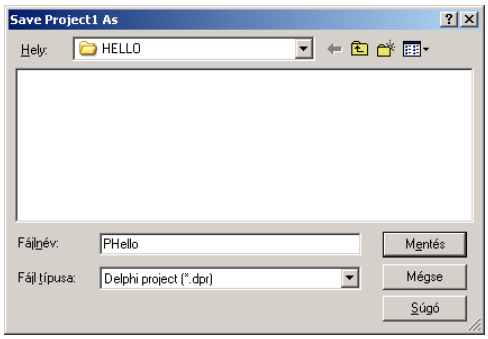

A Delphit tálcára téve

Annak érdekében, hogy ne érhessenek bennünket a későbbiekben meglepetések, célszerű az alkalmazás kezdetén mentést kérni. Zárjuk most be az éppen futó Project1 nevű alkalmazást, melynek legegyszerűbb módja az, ha bezárjuk a Project1 ablakot az X (bezárás gomb) mentén. Ekkor azonnal visszakerülünk a Delphi fejlesztői környezetébe, a Formot újra tervezés alatt fogjuk látni és visszakapjuk az Objektum felügyelőt is.

Ha céltudatosan fejlesztünk, akkor a program indításakor megjelenő ablakban már kérhetjük is a mentést, amit célszerű a Save All funkcióval kezdeményezni, hiszen ekkor kerül mentésre a Unit, a hozzá tartozó Form, valamint a Project is.
Mivel a Delphi fejlesztés során egy project több állományból áll, így minden alkalmazást, amit fejlesztünk önálló könyvtárba célszerű elmentenünk. Ennek érdekében tegyük most a tálcára a Delphit, indítsuk el az Intézőt, és a Hallgató meghajtó Sajátnév könyvtárunkban hozzunk létre egy „DELPHI" alkönyvtárat. Lépjünk be ebbe a könyvtárba, és hozzunk létre egy újabb alkönyvtárat „HELLO" néven. Leendő alkalmazásunk majd egy egyszerű párbeszédablakot fog megnyitni, melyben a szokásos „Helló Világ!" feliratot jelenítjük meg, ezért adjuk most a könyvtárunknak a HELLO nevet. Lépjünk be a HELLO mappába is, majd tegyük a tálcára az Intézőt.
Vegyük fel a tálcáról a Delphit, menjünk fel a File menüre, és válasszuk a Save All lehetőséget (esetleg az eszköztáron lévő gombot megnyomva). Ennek eredménye egy párbeszédablak megjelenése lesz, melyben a Delphi elsőként a Unit mentésére kérdez rá. Mi most a Unitot Uhello néven szeretnénk elmenteni, az előzőleg létrehozott HELLO mappába. Írjuk tehát be a Fájlnévhez: Uhello, (kiterjesztést nem kell adnunk, alapértelmezetten PAS lesz majd) az ablak felső részében pedig a Hely-nél tallózzuk ki az előbb létrehozott HELLO könyvtárat. Ha meggyőződtünk róla, hogy minden rendben, akkor kérjünk mentést:



Mielőtt boldogan hátradőlnénk, látni fogjuk, hogy csupán a Unit mentése nem elég a Delphinek. Mivel a Save All funkciót kezdeményeztük, így most a Projectet is el kell mentenünk. Legyen a neve: Phello, (a kiterjesztése automatikusan DPR lesz) és kerüljön ez is az előző HELLO könyvtárunkba:

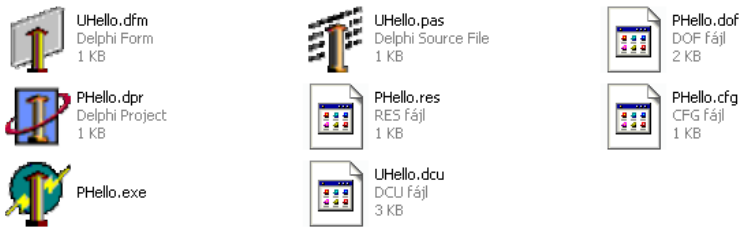


Delphi állományok:

Ellenőrizzük a mentést, vegyük fel a tálcáról az Intézőt, és ha mindent jól hajtottunk végre, látni fogjuk az Uhello.pas, és a Phello.dpr állományokat és még néhány érdekes állományt. A Unitunk mentése során a Delphi két állományt hozott létre a háttérben: az egyik az általunk mentett:

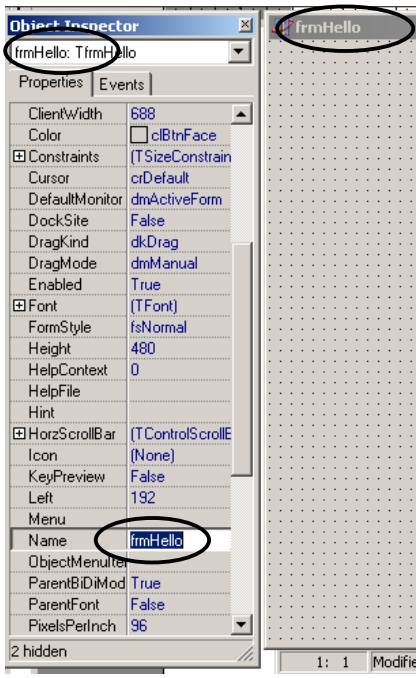
- Uhello.**PAS**, a másik pedig az automatikusan létrejövő
- Uhello.**DFM**, amiről volt már szó korábban: a Formunk kódja. Ezt követően amikor mentettük a Projectet, akkor jött létre maga a
- Phello.**DPR** (a Delphi project állománya), a
- Phello.**DOF** (a projecthez tartozó beállításokat tartalmazza, melyeket a Project/Options almenüből kezdeményezhetünk), a
- Phello.**RES**, (mely a Delphi project számára szóló Windows erőforrás-információkat tartalmazza) és a
- Phello.**CFG** (az aktuális projectünk konfigurációs állománya).

A tevékenységünk során további állományok is létrejönnek majd, például az átmeneti **DCU** állományok, melyek átmeneti állományok, és a még nem mentett változtatásainkat tárolják. Rögtön adódhat a kérdés: de hogyan lesz ezekből futtatható állomány? A válasz roppant egyszerű: futtassuk most újra a már elmentett Delphi munkánkat, majd nézzük meg az intézőt, és látni fogjuk a Projectünk nevével megegyező nevű **EXE** állományt is. Vagyis a Delphi az első futtatás során létrehozza az aktuális könyvtárunkban az exe állományt, amit minden következő futtatáskor felül fog írni.



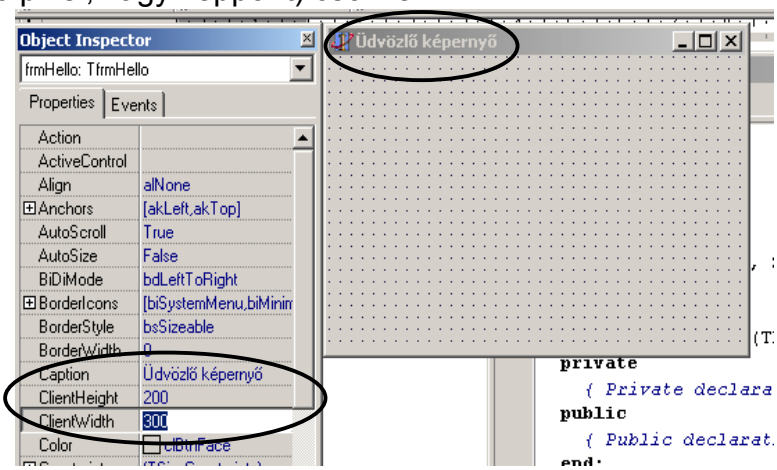
Ezzel az alapok megvannak, az EXE állományunk már önállóan futtatható. Csinosítsunk akkor most a felületen, készítsük el teljesen az alkalmazást.

Térjünk vissza a Delphihez, hozzuk előtérbe a formot, és az objektum felügyelőben győződjünk meg róla, hogy a Formunk (Form1) tulajdonságai láthatóak. Adjunk nevet az űrlapunknak, keressük meg a Properties fülön a Name tulajdonságot, és változtassuk meg: „frmHello”-ra. (Megjegyzés: az elnevezéseknél törekedjünk majd az egységességre, hiszen amennyiben több objektum lesz majd a formon, nehezen fogjuk tudni eldönteni, éppen melyik objektumról is van szó. A későbbiekben néhány tanácsot adunk majd ezzel kapcsolatosan.)



Mint látható, az űrlap nevének változtatása a háttérben megváltoztatja a kódot is, így az objektumkezelőben most már a formunk neve frmHello. Ezzel egyidőben a formunk fejlécében is megváltozott a felirat, amit a következő lépésben a Caption tulajdonság felülírásával állítsunk „Üdvözlő képernyőre”.

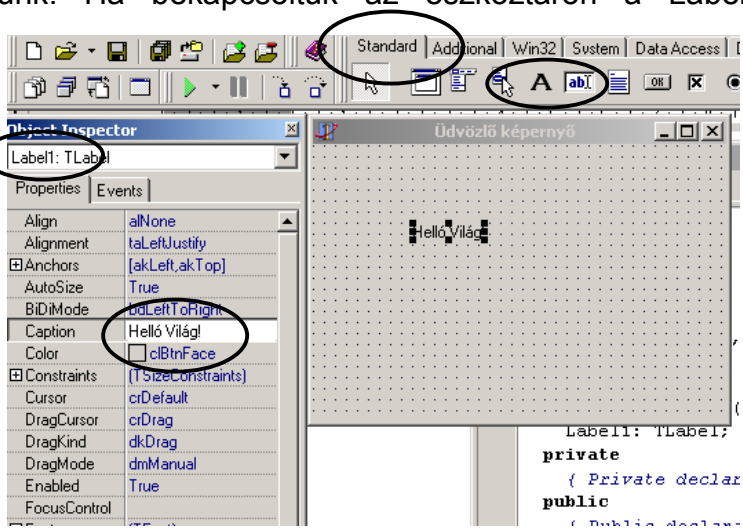
Változtassuk még meg a form méretét is: a Client Height értéke legyen 200, a Client Width pedig 300. Ezzel a form mérete az általunk meghatározott értékre (200*300 pixel, vagy képpont) csökken:



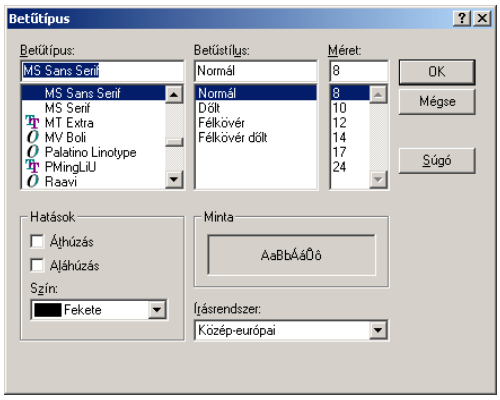
Amennyiben elégedettek vagyunk az eredménnyel, kérjünk mentést a Save All segítségével, és futtassuk az alkalmazásunkat.

Tovább csinosíthatjuk a képet, ha a Caption tulajdonságban szereplő feliratot megpróbáljuk az ablak közepére pozícionálni oly módon, hogy az „Üdvözlő

képernyő” felirat elé szóközöket teszünk.
Mivel a fejlesztés valós időben történik, így a változtatásaink azonnal látszani fognak a képernyőn.
Ideje elhelyezni a tervezett feliratot a formon, amihez a Standard komponenspaletta Label elemét kell kiválasztanunk. Ha bekapcsoltuk az eszköztáron a Label komponenst, vigyük az egeret a formra, és kattintsunk egyet tetszőleges helyen. Megjelenik a Label1 nevű komponens, ennek megfelelően az objektumkezelőben annak tulajdonságai lesznek hozzáférhetőek, aktívak. Keressük ki a Caption tulajdonságot, és Label1-ről írjuk át: „Helló Világ!”-ra:

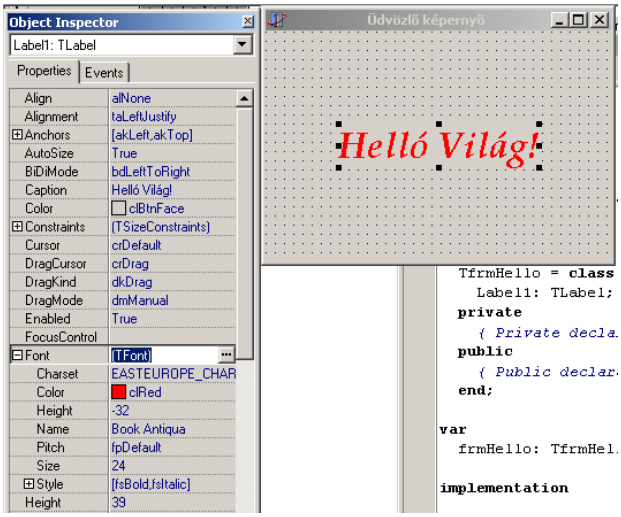


Ha futtatunk, már látszik a kívánt felirat, de még nem igazán tökéletes. Ha szeretnénk beállítani a felirat formai jellemzőit, akkor tervezés alatt az objektumkezelőben a font tulajdonságot kell megkeresnünk. Itt lehetőségünk van a Font felirat előtt elhelyezkedő pluszjelre kattintva megjeleníteni a jellemzőket, amikből aztán lenyíló listák segítségével változtathatjuk meg a szükségesnek ítélteket. Ennek egyszerűbb módjára is lehetőségünk van, ha nem a pluszjelre kattintunk, hanem a font felirat mögött elhelyezkedő … ikonra. (ezt csak akkor fogjuk látni, ha aktívvá tesszük a font tulajdonságot, vagyis ha rákattintunk). Ekkor sokkal barátságosabb módon, egy párbeszédablakban állíthatóak be a szükséges változtatások.



Állítsuk be a következő értékeket:

- Betűtípus: Book Antiqua
- Betűstílus: Félkövér dőlt
- Méret: 24
- Szín: Vörös



Az OK gombbal hagyjuk jóvá a beállításokat, aminek eredményeként a következő képet kellene látnunk:
Természetesen a feliratunk elhelyezkedésén változtathatunk a Drog&Drop módszer segítségével is, de a Delphi képes az objektumok formon belüli elhelyezkedésére egy biztosabb lehetőséget is biztosítani. Jelöljük ki a formon lévő „Helló világ!” címkénket, és kérjük ki a helyi (gyors) menüt, ahol a megjelenő ablakban válasszuk ki az Align… lehetőséget. A megjelenő párbeszédablak magáért beszél, mindkét oszlop elemi közül jelöljük be az utolsó (Center In Window) lehetőséget. Futtassuk az alkalmazást, győződjünk meg róla hogy minden rendben van, ha szeretnénk változtassunk a betűk formai jellemzőin, majd ha befejeztük tevékenységünket, kérjünk mentést, futtassunk egyet utoljára (hisz ekkor íródik felül az EXE állományunk), majd zárjuk be a Delphit. Futtassuk ezek után az intézőből is az EXE alkalmazást önállóan, majd zárjuk be az Intézőt is.

Ha szeretnénk az elkészült munkánkat bemutatni, akkor elegendő egy floppyra az EXE állományt elmentenünk, vagy levélmellékletként elküldeni, és a célgépen futtatni. Amennyiben később még folytatni szeretnénk máshol is a fejlesztést, akkor szükségünk van a PAS állomány(ok)ra, valamint a DFM állomány(ok)ra is. A többes szám itt arra utal, hogy egy project nem csupán egyetlen Unit-ból, és Form-ból állhat. A project állományt (DPR) nem feltétlenül szükséges elmentenünk, mert új projecthez is csatolhatjuk a meglévő PAS, és DFM állományokat, ahogy azt majd a későbbiek során (a többablakos alkalmazások fejlesztésénél) látni is fogjuk.
Szintén az állományokkal kapcsolatban nem árt, ha tudjuk, hogy az elmentett állományainkat az intézőben átnevezve, illetve törölve a Delphi hibaüzeneteket fog generálni, mivel a project állományban tárolt hivatkozásainkat így nem változtathatja meg futásidőben. Összességében törekedjünk arra, hogy módosításainkat mindig a Delphi menüiből kezdeményezzük, lehetőséget adva így a fejlesztői környezetnek arra, hogy munkánkat felhasználóbarát módon, felesleges kódok írása nélkül végezhessük.

Objektum elnevezési szabályok:

Bár az általunk készített munkák nem tartalmaznak túlságosan sok elemet jó, ha már a kezdetekkor hozzászokunk néhány alapvető szabályhoz, melyeknek a későbbiekben fogjuk nagy hasznát venni. Vegyük alapul az elkészült alkalmazásunkat.
A mentésnél célszerű a:

- Unitot: U+munkanév néven elmenteni, vagyis ha a Hello nevezetű Unit-ot mentjük, akkor az állománynév legyen Uhello.pas, ahol az „U” a Unit-ra utal.
- Project: P+munkanév, ahol a „P” utal a Project-re.
- Form neve: frmHello, ahol az „frm” a Form kifejezés rövidítése,
- Label neve: lblHello, ahol az „lbl” utalna a címkére, és így tovább…

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Mi a Delphi nyelvezete?
   a., Object Pascal
   b., Visual Basic
   c., C++
2. Mi az IDE a Delphi környezetben?
   a., Hardvereszközök illesztésének felülete
   b., Integrált fejlesztői környezet
   c., A Delphi interpreteres felülete
3. Mi a BDE?
   a., Komponenskönyvtár
   b., Fordítómotor a Delphihez
   c., Adatbázis adminisztrációs felület
4. Mi a fejrész?
   a., Az űrlap felső része, ahol az alkalmazás neve is megtalálható
   b., A menüt, eszköztárat, és komponenseket tartalmazó felület
   c., Az objektum felügyelő felső része.
5. Hány csoportra oszlik a komponenspaletta?
   a., 9
   b., 12
   c., 15
6. Melyik billentyű lenyomására jelenik meg a kódszerkesztő ablaka?
   a., F8
   b., F10
   c., F12
7. Mi az SDI?
   a., Több ablakos alkalmazás fejlesztésének lehetősége.
   b., Egy ablakos alkalmazás fejlesztésének lehetősége.
   c., Shut Down Interface.
8. Melyik menüből érhető el a „Use Unit” menüpont?
   a., File.
   b., Project.
   c., Compile.

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. A Delphi project állománya a DFM kiterjesztésű állomány.
   igaz
   hamis
2. Az alapértelmezett alkalmazásnév a Delphiben FORM1.
   igaz
   hamis
3. A Delphi fejlesztését megelőzte a Microsoft Visual Basic.
   igaz
   hamis
4. A Delphi telepítésének része az Interbase telepítése is.
   igaz
   hamis
5. A Delphi fejlesztőkörnyezet négy fő részből áll.
   igaz
   hamis
6. A Delphi menüi helyzetérzékenyek.
   igaz
   hamis

III.

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Hol található meg a Lock Controll menüpont?
2. Hány csoportot tartalmaz a komponenspaletta?
3. Soroljon fel 5 elemet a Project menüből!
4. Sorolja fel a Delphi állományait, azok kiterjesztését!
5. Sorolja fel a fejlesztői környezet 4 csoportját!
6. Ismertesse a mentés folyamatát Delphiben!
7. Mi az Object Inspector?
8. Sorolja fel az eszköztár legalább 5 elemét!

Objektumok kezelése

Project megnyitás, létrehozás

Megnyitás:

Az előző alkalommal egy példafeladatot készítettünk el, melyet a HELLO könyvtárunkba mentettünk. Szeretnénk folytatni a munkát, így vissza szeretnénk kapni a korábban elmentett fejlesztői környezetet. Ha a teljes munkát szeretnénk visszahozni, akkor több lehetőség közül választhatunk.

1. Keressük meg az intézőben a könyvtárunkat, majd az ott lévő Phello.dpr állományra kattintsunk duplán.
2. Indítsuk el a Delphit, majd a file menüben keressük meg a Reopen menüpontot. Ha nem dolgoztak mások a Delphivel, akkor az első állomány az lesz. Vigyázzunk arra, hogy a project állományt indítsuk el, ugyanis a Unit önmagában nem futtatható, csupán a forrás módosítását teszi lehetővé.
3. Ha mások is dolgoztak a gépen, akkor elképzelhető, hogy nem találjuk a Reopen menüben korábbi állományunkat. Ekkor válasszuk a File menü Open Project lehetőségét, és tallózzunk a megfelelő könyvtárba, majd válasszuk ki a projectet.

Ellenőrzésképpen célszerű futtatni is rögtön az alkalmazást, mert jobb, ha már most kiderül, hogy nem működik valami, mert letöröltünk néhány fontos állományt…

Project létrehozása:

Természetesen még ekkor sincs elveszve minden. Ha a Unit, és a DFM állományok megvannak, azokból bármikor tudunk újra projectet létrehozni, aminek menete a következő:

- Töröljünk le minden állományt a HELLO könyvtárból, az Uhello.pas, és az Uhello.dfm állományok kivételével.
- Indítsuk el a Delphit, majd menjünk a Project menüpont Add To Project almenüjére. Tallózzuk ki a megjelenő párbeszédablakban a Hello könyvtárat, és adjuk hozzá a projecthez az Uhello.pas állományt. A képernyőn azonnal meg fog jelenni a Form is, ezért nekünk nem kell semmit sem tennünk.
- Van még egy apróság: a Delphi indulásakor létrejött egy Unit1 nevű Unit, amire nekünk most semmi szükségünk. Menjünk fel a Project menüpont Remove From Project almenüjére, és a megjelenő párbeszédablakból távolítsuk el a Unit1 nevű Unit-ot. Ezzel a hozzá tartozó form is megsemmisül.
- Egyetlen dolgunk maradt: kérjünk mentést, a Save All funkcióval. Ekkor a Unit-ot már nem kell mentenünk, hiszen azt korábban megtettük, csupán az új project állományunknak kell nevet, és elérési utat beállítanunk. Legyen a neve: Phello.dpr, és kerüljön a HELLO könyvtárba.

Ezt a módszert bátran alkalmazhatjuk arra is, hogy az iskolai munkát otthon folytassuk, hisz a fenti két állomány mérete jelentősen kisebb, mint a könyvtárunkban található egyéb állományok. Természetesen a több formból álló alkalmazásaink fejlesztése során valamennyi Unit, és DFM állományt haza kell majd vinnünk, de a project ekkor is csak egyetlen állomány.

A formokról általában:

A Delphi alapeleme a form vagy űrlap. Az űrlap komponens neve TForm. A Delphi indításakor kapott üres ablak egy TForm komponens, amelyen pontozott rács helyezkedik el. A rács segíti a szükséges komponenseket elhelyezni az űrlapon. Az alkalmazás készítésének első szakasza nem más, mint a TForm objektum megszerkesztése (a komponensek elhelyezése), a tulajdonságok és eseménykezelések megadása.

Objektumok tulajdonságai:

Ha elindítottuk a Hello projectet, akkor alapértelmezetten a Formunk (frmHello) tulajdonságait látjuk az objektum-kezelőben. Mint korábban szó volt róla, az objektumoknak eltérő tulajdonságaik lehetnek, így egy adott objektum (mondjuk űrlap) tulajdonságlapját összehasonlítva egy másik objektum (mondjuk címke) tulajdonságaival más és más tulajdonságokat fogunk látni. Ebben a fejezetben igyekszünk áttekinteni a legjellemzőbb tulajdonságokat a teljesség igénye nélkül a form (mint az egyik legtöbb tulajdonsággal rendelkező) objektumot alapul véve. Egyéb esetlegesen ismeretlennek tűnő tulajdonság esetén elég, ha kijelöljük a tulajdonságot, majd megnyomjuk az F1 billentyűt.

ActiveControl

Azt határozhatjuk meg, hogy melyik komponensen legyen a fókusz, azaz pl. gomb komponensek esetén melyik nyomógomb legyen az alapértelmezett (körülötte sötétebb keret). A tulajdonság neve melletti, egyelőre üres részre kattintva, a legördülő listából - amely a formon található komponenseket tartalmazza - kiválaszthatjuk a kívánt elemet.

AutoScroll

A tulajdonságnak két állása van: True és False. A True beállítása azt jelenti, hogy az ablak - amennyiben futási időben olyan kicsire módosítódik, hogy tartalma nem fér el benne - automatikusan kiegészül gördítősávval.

BorderIcons

A tulajdonságnevek egy része előtt + jel látható (hasonló a Windows Intézőhöz). Ez azt jelenti, hogy ha a jelre rákattintunk, akkor további altulajdonságok jelennek meg, amelyeket egyenként lehet beállítani (ha - jel van előtte, akkor viszont összezárható). A BorderIcons tulajdonsággal egyébként a Windowsos rendszermenü és ablakméretező gombok megjelenéséről rendelkezhetünk.

- biSystemMenu: a rendszermenü (vezérlőmenü) engedélyezése vagy tiltása
- biMinimize: az ablakminimalizáló gomb engedélyezése.
- biMaximize: az ablakmaximalizáló gomb engedélyezése.
- biHelp: a Help gomb engedélyezése.

BorderStyle

Az ablakszegély stílusát és méretezhetőségét határozza meg. A legördülő listából a következők közül választhatunk:

- bsDialog: szabványos párbeszéd-ablak keret, nem méretezhető.
- bsSingle: nem méretezhető szimpla vonalas keret.
- bsNone: nem látható, nem méretezhető keret, ablakmaximalizáló, minimalizáló ikonja, vagy rendszermenüje lehet.

- bsSizeable: normál méretezhető keret.
- bsSizeToolWin: olyan mint a bsSizeable, de kisebb címsora van.
- bsToolWindow: olyan mint a bsSingle, de kisebb címsora van.

Caption

Magyarul: képszöveg, felirat. Itt az ablak címsorában szereplő szöveg alapértelmezetten az ablak neve lesz. A mezőbe közvetlenül gépelhetjük be a kívánt feliratot.

ClientHeight, ClientWidth

A form hasznos, kereten belüli méretét lehet megadni pixelben.

Color

A form háttérszínét adhatjuk meg. Vagy a legördülő listából, vagy a jobb oldali cellára kétszer kattintunk, s a megjelenő párbeszéd-ablakból választhatunk.

Cursor

Rengeteg kurzorforma közül válogathatunk, amely a form fölötti helyzetre értendő. A súgóban meg lehet tekinteni.

Enabled

A form csak True állásban reagál külső eseményekre, False állapotban nem, vagyis itt adható meg hogy engedélyezzük a hozzáférést vagy sem.

Font

A betűk megjelenítését határozhatjuk meg egy többszintű beállítási listán, amit az előző alkalommal már részletesen áttekintettünk.

FormStyle

Négy beállítási lehetőség közül választhatunk. Az

- - fsNormal az alapértelmezett, az
- - fsStayOnTop esetén az ablak mindig a többi fölött fog elhelyezkedni, az
- - MDI-s tulajdonságokat pedig MDI formok esetén kell választani.

Height

A form magasságát állíthatjuk be pixel egységben.

HelpContext

Súgó alkalmazása esetén használatos, a súgó környezetazonosítót tartalmazza.

Hint

Ez is a súgóval kapcsolatos tulajdonság (csak a ShowHint tulajdonság True-ra állításánál fog működni).

HorzScrollBar

A függőleges gördítősáv beállítását adhatjuk itt meg (a láthatóságot a Visible tulajdonság True-ra állításával biztosíthatjuk). A vízszintes gördítősáv tulajdonságait a VertScrollBar-al határozhatjuk meg.

Icon

A kívánt ikont a három pontra kattintva kereshetjük meg. Az ikon a rendszermenü jeleként fog szerepelni, s ezt látjuk a Tálcán összezárt alkalmazásikonon is.

KeyPreview

A billentyűleütéseket a False beállításkor a komponens, True választásakor pedig a form dolgozza fel.

Left

A form helyzetét állíthatjuk be a képernyő bal oldalától (pixelekben mérve). A felső résztől való távolságot a Top tulajdonság írja le.

Menu

Menütervezésnél használjuk. Külön lesz még róla szó.

Name

A formnak (általában az objektumoknak) adhatunk nevet, hogy olvasható programszövegünk legyen. Legfeljebb egyszerű programoknál hagyjuk meg az eredetit. A név- (azonosító-) adás szabályai már a Pascal-ból is ismertek, de az előző fejezet végén is volt szó róla.

ObjectMenuItem

OLE objektum menücímként való kezelésekor használjuk.

PixelsPerInch

Azt állíthatjuk be, hogy a form egy inch-nyi részén hány képernyőpont (pixel) legyen (a Scaled tulajdonság True-ra állításánál). A képernyőfelbontástól független programírást teszi lehetővé.

PopupMenu

A helyi vagy gyorsmenü nevét adhatjuk meg.

Position

A form elhelyezését leíró tulajdonság. Öt beállítási lehetőség van:

- poDefault: teljes mértékben a Delphi dönt a form helyéről és méretéről.
- poDefaultPosOnly: a Delphi dönt a form helyéről, méretét viszont nem változtatja.
- poDefaultSizeOnly: a form ott lesz, ahol a tervezéskor hagytuk, de a méretet a Delphi állítja be.
- poDesigned: ott és olyan lesz a form, amilyen a tervezéskor volt.
- poScreenCenter: a képernyő közepére helyezi a formot.

PrintScale

A form kinyomtatási módját leíró tulajdonság.

- poNone: nem lesz léptékváltoztatás, így a nyomat különbözni fog a képernyőtől.
- poPrintToFit: a nyomtatási méretek úgy módosulnak, hogy ráférjen a papírra.
- poProportional: a képernyőn és a papíron lévő form méretei megegyeznek.

Scaled

A PixelsPerInch tulajdonságnál volt róla szó.

ShowHint

A Hint tulajdonságnál volt róla szó.

Tag

Egész számot adhatunk meg, amivel információt tárolhatunk az adott objektumról.

Top

A form helyzetét állíthatjuk be a képernyő tetejétől (pixelekben mérve), mint a korábbi Left tulajdonság esetén.

VertScrollBar

Lásd a HorzScrollBar-nál leírtakat.

Visible

A form látható vagy sem (futási időben).

Width

A form szélességi méretét lehet beállítani pixelekben.

WindowMenu

MDI alkalmazásoknál ablakmenüt lehet a formhoz rendelni.

WindowState

A form alapértelemezett méretarányát leíró tulajdonság.
wsMaximized: Maximalizált (teljes képernyő) méretű form.
wsMinimized: Minimalizált méretű (tálcán megnyíló) form.
wsNormal: Sem nem maximalizált, sem nem minimalizált méretű form.

Eseményvezérelt eljárások:

Ahogy az objektumoknak eltérő tulajdonságaik vannak, ugyanúgy eltérőek lesznek a hozzájuk kapcsolódó eseményvezérelt eljárások is. Nézzük ismét a form objektumot, milyen események tartoznak hozzá. (Itt is érvényes a korábban említett kijelöléssel történő súgó aktiválás az F1 billentyűvel)
A Delphi programok eseményvezéreltek, ami azt jelenti, hogy a program valamilyen esemény bekövetkezésére (pl. egy nyomógomb megnyomására, billentyűleütésre stb. reagálva) fut tovább. Az események kezelését segíti az objektum-felügyelő második, Events nevű lapja. Az eseménykezelést lényegében nekünk kell megírnunk, amihez a Delphi forráskódú sablont szolgáltat. A sablont az eseménynév jobb oldali cellájára való kétszeri kattintással hívhatjuk elő. Az alapértelmezett eseményeket a komponensre történő dupla kattintás teszi elérhetővé.

OnActivate

Akkor hajtódnak végre az itt található utasítások, amikor az inputfókusz áttevődik a formra.

OnClick

A formra való egérrel történő kattintás váltja ki.

OnClose, OnCloseQuery

Az OnCloseQuery eseményt az ablak bezárása, pontosabban bezárási kísérlete váltja ki, és az OnCloseQuery esemény végrehajtása után záródik majd be a form. A formot bezárhatjuk az Alt+F4 billentyűpárossal, a bezáró ikonnal stb. Használhatjuk az eseményt a form bezárásának letiltására, memória-felszabadításra. Az OnClose esemény határozza meg az ablakbezárás módjait (az esemény Action paraméterei: caNone, caHide, caFree, caMinimize).

OnCreate

A form és komponensei kezdeti tulajdonságainak beállítására használjuk. Az esemény a form első végrehajtásakor következik be.

OnDblClick

A Formon történő kétszeres egérkattintás váltja ki.

OnDeactivate

Más alkalmazásra való váltáskor következik be (pl. a programot a Tálcára tesszük alkalmazásikon formájában, s egy másik alkalmazással, pl. szövegszerkesztővel kezdünk dolgozni).

OnDestroy

A memória-felszabadítás eszköze ablakbezáráskor.

OnDragDrop, OnDragOver

Az egér mozgását figyelő események.

OnHide

Az ablak elrejtésekor következik be.

OnKeyDown, OnKeyPress, OnKeyUp

A billentyűleütéseket figyelő események.

OnMouseDown, OnMouseMove, OnMouseUp

Az egér mozgását figyelő események.

OnPaint

Ha az ablak egy részét ideiglenesen eltakarta valami (pl. egy másik ablak), akkor az újrarajzolásról itt kell gondoskodni.

OnResize

Az ablak átméretezésekor az elemek áthelyezését és átméretezését itt kell megoldani.

OnShow

A form megjelenése előtt bekövetkező esemény.

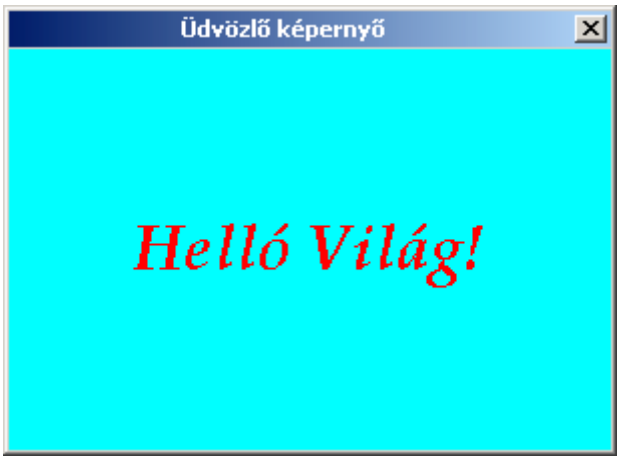
Űrlap objektum tulajdonságainak módosítása:

A tulajdonságok beállításának gyakorlása végett végezzük el a következő feladatokat a Hello munkánkon:

- Kattintsunk a Form szürke üres területére, és győződjünk meg róla, hogy az objektum-felügyelőben a felső részben az „frmHello” felirat látszik.

- A Border Icons tulajdonságoknál el tudjuk tüntetni a kisméret, és a teljesméret gombokat, ha a biMinimize, és biMaximize értékeket False-ra állítjuk.
- Ahhoz, hogy megakadályozzuk az űrlapunk átméretezhetőségét, állítsuk át a Border Style tulajdonságot bsDialog-ra
- Válasszuk ki a Color tulajdonságot, és kattintsunk duplán az ott látható szín feliratán. A megjelenő párbeszédablak ismerősnek tűnhet, válasszunk egy színt, vagy definiáljunk saját színt az egyéni színek definiálása gomb segítségével. Legyen a választott szín most halványkék.
- Állítsuk a Position tulajdonságot poScreenCenter-re, vagyis ablakunk a képernyő közepén nyíljon majd meg.

Mentés, és futtatás után amennyiben mindent megfelelően állítottunk be, a következő képet kellene kapnunk:



A beállítások eredményeként az ablakot nem lehet a keret mentén méretezni, nem nyitható teljes méretre, és nem tudja a felhasználó a tálcára tenni. Ezen kívül a képernyőnk közepén fog megjelenni.

Kódszerkesztés a Delphiben:

A Windows egész filozófiája az ablakokra van alapozva, ezért a Windowsos felületre tervezett Delphi alkalmazásokat is ablakokban (formokban) kell elképzelni. A program (project) különböző részei, műveletei egy-egy, arra a feladatra tervezett ablakban játszódnak le. Magyarul: a program felhasználói felületét az ablakok szolgáltatják.
A Delphiben kétféle programozási stílussal dolgozhatunk. Az egyik a komponensekkel való programozás, aminek az alapjain már lassan túljutunk. A komponensek előre elkészített építőelemek, amelyekből úgy állítjuk össze a programot, mint egy Lego-játékot az építőkockáiból. A másik programozási mód a "hagyományos": a problémamegoldás Object Pascal programnyelven való megfogalmazása. A Delphi a forráskód egy részét saját maga készíti el, a tulajdonképpeni algoritmusokat nekünk kell megírnunk.
De mik is a komponensek? Adott feladatra tervezett elemek, amelyeket Object Pascal-ban írtak meg, s amelyek forráskódjához a Delphi bizonyos verzióiban hozzá is férhetünk. A komponenseket a nevük elé írt T betűről ismerhetjük fel. A komponenseket az űrlapon (formon) helyezhetjük el. Maga a form komponens neve TForm, a ráírt szöveg (címke) TLabel, a ráhelyezett nyomógomb pedig TButton stb.
Ha eseményeket szeretnénk rendelni az objektumainkhoz, meg kell keresnünk az objektum-kezelőben az objektum Events fülén lévő valamely eseményt, és végre

valóban „programozhatunk”, a korábban már elsajátított Pascal utasításainkra alapozva.

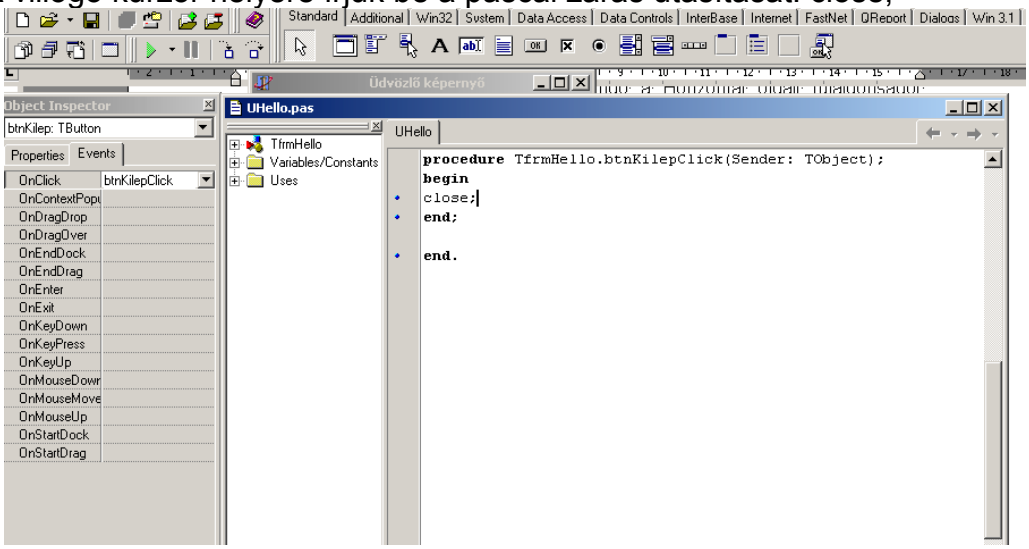
Parancsgomb használata:

Nézzük a már egyre látványosabb Hello projectünket. Szeretnénk egy picit aktívabbá tenni a dolgot, mondjuk, a kilépéshez feltennénk egy gombot a felhasználóknak.

- Tervező nézetben keressük meg a Standard komponenspalettán, középtájon az OK feliratot viselő BUTTON objektumot, és kattintsunk rá (jelöljük ki).
- Vigyük az egeret a Helló Világ! feliratunk alá, és kattintsunk egyet a Form területén.
- A megjelent gombunk tulajdonságait állítsuk be:
    - Caption: &Kilépés (az & jel a gyorsbillentyűvel történő elérést szolgálja, így a feliratunkban az első karakter aláhúzott lesz futásidőben)
    - Name: btnKilep (a korábban már megbeszélt névkonvenciók alapján)
    - A gombon jobb egérgombbal kattintva válasszuk ki az Align tulajdonságot, és most elegendő a Horizontal oldali tulajdonságot Center In Window-ra állítani.

Ha a mentés után futtatunk, már látni fogjuk a megjelenésében teljes értékű gombunkat, melynek egyetlen szépséghibája, hogy még nem tesz semmit, ha rákattintunk. Ezt az apró problémát gyorsan kiküszöbölhetjük:

- Menjünk tervező nézetbe, a formon válasszuk ki egy kattintással a gombot.
- Győződjünk meg róla, hogy az objektum-kezelőben a btnKilep tulajdonságait látjuk, majd kattintsunk az Events fülre.
- A gombok esetében az alapértelmezett (első) esemény az OnClick, vagyis a kattintásra esemény. Itt, ha megnézzük, a lenyíló lista nem kínál választási lehetőséget, de mi nem is szeretnénk ilyesmit. Kattintsunk duplát az aktív mezőbe (vagyis az OnClick esemény mellett lévő listába) és azonnal megjelenik a kódszerkesztő.
- A villogó kurzor helyére írjuk be a pascal zárás utasítását: close;



- Kérjünk mentést, és futtassuk az alkalmazást.

Amennyiben mindent jól hajtottunk végre, a gombunk működni fog, vagyis ha rákattintunk, be fogja zárni az ablakot.

A Unit tartalma

Természetesen nem mehetünk el szó nélkül az előbb látott ablak (a kódszerkesztő) mellett, értékeljük ki a tartalmát. Tervezőnézetben ha nem a Unit ablak lenne az aktív, akkor hívjuk azt elő.
A görgetősáv, és az ablak méretezésének segítségével igyekezzünk láthatóvá tenni a teljes kódot:

```
unit UHello;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TfrmHello = class(TForm)
    Label1: TLabel;
    btnKilep: TButton;
    procedure btnKilepClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmHello: TfrmHello;

implementation

{$R *.DFM}

procedure TfrmHello.btnKilepClick(Sender: TObject);
begin
close;
end;

end.
```

Nos, ahhoz képest, hogy mindebből mi csupán a „close” utasítást írtuk be, itt elég sok minden található. Akik jártasak a Pascal nyelvezetében, azok számára nem kell mélyen magyarázni a fent látható kódot. Érdekességként annyit emelnénk ki, hogy az általunk névvel el nem látott címke az eredeti „delphi konvenció” szerint az osztályának a nevét, és a létrehozás során kapott sorszámot kapta: Label1. Amennyiben átnevezzük az objektum-kezelővel az objektumot, akkor természetesen a háttérben a kód automatikusan megváltozik.
Ebből is érzékelhető talán a Borland fejlesztőinek célja: egy játékos, egyszerűen használható fejlesztői környezet elkészítése, mely a lehető legnagyobb mértékben

felhasználóbarát, és lehetőséget ad a komolyabb célokkal programozni kívánók számára is.
Játszunk még el egy kicsit a Hello kóddal, hajtsuk végre a következő feladatokat:

- Hívjuk elő a háttérből a formunkat (frmHello)
- Kattintsunk a form világoskék felületére
- Az objektumkezelőben a form Events fülén kattintsunk duplán az OnClick esemény beviteli mezőjébe.
- A megjelenő begin end páros közé írjuk be:

```
Label1.Caption:='Hello World!';
```

Ez az apró kódsorozat azt eredményezi, hogy futás közben az űrlapra kattintva (természetesen nem a bezárás gombra gondoltunk), megváltozik a feliratunk szövege.
Ebben az egy utasítássorban két dologra is érdemes felfigyelnünk. Az egyik, hogy objektumaink tulajdonságait futásidőben a kódon keresztül tudjuk befolyásolni. (Ezt még annyiban segíti a Delphi, hogy a kód helyes írásakor az objektumokhoz megjeleníti a hozzájuk tartozó megváltoztatható tulajdonságot is. Amennyiben figyeltük a képernyőt, miközben az előző kódot beírtuk ezt már tapasztalhattuk is.). A másik a szöveges információk szimpla idézőjelek közé tétele, vagyis a címkénk szöveges (text) típusú adatot fogad, amit az értékadáskor szimpla idézőjel nyit, és ugyanaz zár.
A Pascal utasítások lezárása itt is érvényes, vagyis az utasításokat pontosvesszővel zárjuk, de a Delphi nem kényes annyira az esetlegesen elhagyott jelekre.

Befejezésként hajtsuk végre önállóan azt a feladatot, hogy amennyiben a feliratunkra kattintunk futásidőben, akkor az eredeti feliratunk térjen vissza, vagyis újra a „Helló Világ!” feliratot lássuk.

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Melyik két állomány szükséges legalább korábbi munkánk folytatásához?
   a., DPR, PAS
   b., DFM, DPR
   c., PAS, DFM
2. Melyik állomány tartalmazza az általunk írt kódot?
   a., DFM
   b., DPR
   c., PAS
3. Melyik objektumtulajdonság alkalmas a vezérlőmenü gombjainak beállítására?
   a., Border Style
   b., Object Menu Item
   c., Border Icons
4. Minek a beállítására alkalmas a Window State?
   a., Az ablak elhelyezkedését befolyásolja a képernyőn.
   b., Az ablak méretét adhatjuk meg pixelben.
   c., Az ablak méretarányát állíthatjuk be.
5. Mikor hajtódik végre az OnActivate utasítások?
   a., Ha elindítjuk a formot.
   b., Ha betöltődik a form.
   c., Ha a fókusz a formra kerül.
6. Hányféle programozási stílussal dolgozhatunk a Delphiben?
   a., 2.
   b., 3.
   c., 5.
7. Hová kerülnek a kódban a komponensek definíciói?
   a., a USES részbe
   b., a TYPE deklarációs részbe
   c., az IMPLEMENTATION után
8. Hogyan nevezi el az objektumokat alapértelmezetten a Delphi?
   a., osztálynév+típus
   b., osztálynév+növekvő sorszám osztályonként
   c., osztálynév+az objektum formon levő sorszáma

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. A Delphi nem érzékeny arra, ha elhagyjuk a sor végén lévő lezáró jelet.
   igaz
   hamis
2. Az objektumoknak azonos tulajdonságaik vannak.
   igaz
   hamis
3. Minden objektumhoz tartozik OnActivate esemény
   igaz
   hamis
4. Az objektumosztályok betűjelzése a T.
   igaz
   hamis
5. Az űrlap átméretezhetőségét a Border Style tulajdonsággal szabályozhatjuk.
   igaz
   hamis
6. Az OnShow esemény az űrlap megjelenésekor kerül végrehajtásra.
   igaz
   hamis
7. A HorzScrollBar a vízszintes görgetősáv tulajdonságainak beállítására szolgál
   igaz
   hamis
8. A Height tulajdonság a Form magasságának beállítására szolgál.
   igaz
   hamis
9. Korábbi delphi munkánkat a Reopen paranccsal biztosan elő tudjuk hívni.
   igaz
   hamis

III.

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Mit tekintünk a Delphi objektumok közül alapobjektumnak?
2. Soroljon fel legalább 5 beállítható objektum tulajdonságot!
3. Soroljon fel legalább 5 objektumokhoz kapcsolódó eseményt!
4. Mit állíthatunk be a BorderStyle tulajdonsággal?
5. Mikor aktivizálódik az OnKeyUp esemény?
6. Vázolja fel a háttérben lévő UNIT tartalmát!

A Delphi komponensei I.

A komponenspaletta

Az előzőekben már megismerkedtünk néhány komponenssel (Button, Label), eljött az idő, hogy ismereteinket tovább mélyítsük. A Delphi játékos fejlesztési felületét leginkább az előre elkészített komponensek használata adja. Ha egy gombra van szükségünk, már nem nekünk kell megírnunk a gombot generáló forráskódot, mert ezt már elvégezték helyettünk. Nekünk csak ki kell választanunk azt, amelyikre éppen szükségünk van. Természetesen ez nem jelenti azt, hogy nincsen lehetőségünk önálló, saját fejlesztésű komponenseket definiálni. De nézzük tehát, mik azok az elemek, melyeket választhatunk:

Standard komponenspaletta:

Az egyik leggyakrabban használt csoport, itt találhatóak meg az alapvető komponensek.
Elemeit balról jobbra soroljuk fel:

- Nyíl: a legelső aktív elem, az objektumok kiválasztását teszi lehetővé.
- Frames: frame-eket hozhatunk létre (ablak az ablakban).
- MainMenu: Menüszerkezet létrehozását teszi lehetővé.
- PopupMenu: Helyi, vagy gyorsmenü létrehozásának lehetősége.
- Label: Feliratok, címkék elhelyezésére szolgál.
- Edit: Beviteli mezőt hozhatunk létre.
- Memo: Feljegyzést készíthetünk (hosszabb szövegek tárolására).
- Button: Nyomógomb.
- CheckBox: Jelölőnégyzet (csoportba foglalható).
- RadioButton: Választókör (csoportba foglalható).
- ListBox: Listapanel, elemeket sorolhatunk fel benne.
- ComboBox: Lenyílólista.
- ScrollBar Görgetősáv.
- GroupBox: A csoportba foglalás eszköze.
- RadioGroup: Választókörök csoportja.
- Panel: Egyéb objektumok egységbe foglalását teszi lehetővé.
- ActionList: Eljárások, függvények, akciók tárolására szolgál.

Valamennyi elem esetében érvényes, hogy kiválasztásuk után kattintanunk kell a formon, így lesznek azok a form részei. A fenti csoportból a MainMenu, a PopupMenu, és az ActionList futásidőben láthatatlan, tehát a formunk bármely területére elhelyezhetőek. Az általuk végrehajtandó tevékenység ugyanis külső segítséggel történik majd meg, például a menü esetében mi csupán a menüelemeket kell, hogy felsoroljuk, és a hozzájuk tartozó utasításokat, a megjelenítés a Windows konvencióinak megfelelően történik.

Feladat: Másolás (Masol)

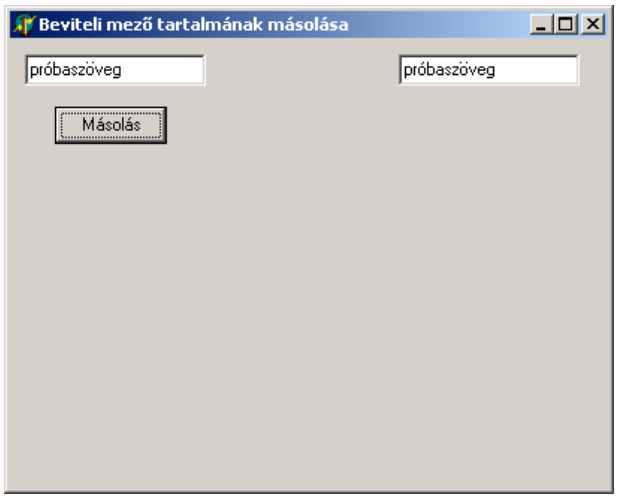
Készítsünk egy egyszerű alkalmazást, mely egy beviteli mezőbe írt szöveget egy gomb megnyomására egy másik beviteli mezőben is megjelenít.

1. Indítsuk el az Intézőt, és hozzuk létre a Delphi alkönyvtárunkon belül a Masol könyvtárat, ahová a munkánkat fogjuk menteni.
2. Indítsuk el a Delphi-t, majd az alapértelmezett Form1 tulajdonságait változtassuk meg a következőképpen:
   - Name: frmMasol.
   - Caption: Beviteli mező tartalmának másolása.
   - ClientHeight: 300
   - ClientWidth: 400
3. Kérjünk mentést a Save All utasítással. A Unit neve legyen Umasol.pas, a project neve pedig Pmasol.dpr. Mindkét állomány a korábban létrehozott Masol könyvtárunkba kerüljön.
4. Keressük meg a Standard komponenspalettán az Edit elemet, kapcsoljuk be, majd helyezzük azt a formra, a bal felső sarokba. Mivel ez az aktív elemünk, végezzük el rajta a beállításokat az objektum-felügyelő segítségével:
   a. Name: edtForras
   b. Text: Töröljük ki a mező tartalmát. Ha be van ide írva valami, akkor az alapértelmezetten megjelenik futásidőben, ami nem mindig szerencsés.
   c. Top: 10 (Pixelben megadjuk az elhelyezkedést).
   d. Left: 10 (szintén az elhelyezkedést állítja be).
5. Válasszuk ki újra a komponenspalettán az Edit elemet, és helyezzük a már meglévő elemünk mellé. Állítsuk be ennek is a tulajdonságait:
   a. Name: edtCel
   b. Text: Töröljük ezt is.
   c. Top: 10
   d. Left: 260 (Ezt közelítőleg adtuk meg, ismerve a Form szélességét (300), és ismerve az Edit alapértelmezett szélességét (21))
6. Az alapelemeink megvannak, válasszuk ki a komponenspalettáról a másolás végrehajtásáért felelős Button gombot, és helyezzük a forrást tartalmazó beviteli mező alá. Állítsuk be a tulajdonságait:
   a. Name: btnMasol
   b. Caption: Másolás
   c. Left: 30
   d. Top: 45
7. Írjuk meg a gombhoz tartozó kódot. Az előző alkalommal már írtunk hasonló utasítást, akkor részletesen jutottunk el a kódszerkesztőhöz, lépésről lépésre. Most ezt egyszerűbben tesszük meg: kattintsunk duplán a bal egérgombbal a Másolás gombra. Látni fogjuk, hogy a gomb alapértelmezett eseményével indul el a kódszerkesztő:

```
procedure TfrmMasol.btnMasolClick(Sender: TObject);
begin
|
end;

end.
```

Beviteli mező tartalmának másolása:

8. Mi azt szeretnénk, ha a gomb lenyomásakor a gép, a forrásban található szöveget átmásolná a cél beviteli mezőbe. Ennek érdekében a villogó kurzor helyére írjuk be a következő kódsorozatot: **edtCel.text:=edtForras.text;**
9. Kérjünk mentést a Save All-al, majd futtassuk és teszteljük az alkalmazásunkat.



Bár az eredeti célt megvalósítottuk, vagyis a feladat megfogalmazásában szereplő tevékenység maradéktalanul végrehajtásra kerül, szemmel látható, hogy programunk hagy némi kívánnivalót maga után.

1. A tesztelés során kényelmetlen a másolás végrehajtása után újra rákattintani a forrásra, és átírni annak tartalmát.
2. A cél mező hozzáférhető, így a felhasználó át tudja írni annak tartalmát.
3. Bár nem okoz hibát, de értelmetlen művelet, ha a forrás mező tartalmát üresen másoltatjuk át a cél mezőbe.

Ezeknek a hiányosságoknak a kiküszöbölése érdekében bővíteni fogjuk a programot a következőképpen:

A fókusz vezérlőelemre irányítása:

10. Az első problémára megoldást jelenthet, ha kiegészítjük az előzőleg beírt kódot egy újabb sorral. Keressük meg tervezőnézetben a Másolás gombot, és kattintsunk rá duplán. Az előzőleg beírt sor végén üssünk egy Entert, és új sorba írjuk be**: edtForras.SetFocus;** Ezzel az utasítással a másolás befejezése után arra utasítjuk a programot, hogy a fókusz kerüljön vissza a forrás beviteli mezőre. Természetesen így a forrásban lévő szöveg kijelölésre kerül majd (sötétkék lesz a háttere), ami nekünk tökéletes, hiszen így rögtön felülírhatjuk az új szöveggel a tartalmát.
11. A második probléma kiküszöbölésének érdekében tervezőnézetben válasszuk ki a formon az edtCel mezőt, és keressük meg a tulajdonságlapon a ReadOnly (csak olvasható) tulajdonságot, majd állítsuk True-ra. (Elvileg az Enabled (engedélyezve) tulajdonsággal is játszhatnánk, de ebben az esetben a másolt szövegünk szürkés színnel jelenne majd meg a formon a cél mezőben, ami nem túl szép.)
12. A harmadik probléma megoldásához már összetettebb kódra lesz szükségünk. Egy elágazásban meg kell vizsgálnunk a másolás megkezdése előtt a mező tartalmát, és ha üres, akkor nem hajtjuk végre a másolást. Ehhez újra vissza kell térnünk a gombunk már módosított kódjához, kattintsunk

duplán a Másolás gombra tervezőnézetben. A megjelenő kódot írjuk át a következőképpen:

```
procedure TfrmMasol.btnMasolClick(Sender: TObject);
begin
  If edtForras.Text='' then
  Else
        edtCel.text:=edtForras.text;
        edtForras.SetFocus;
end;

end.
```

Egyszerű üzenet küldése:

Próbáljuk ki az alkalmazásunkat, mentsünk, és futtassunk. Látni fogjuk, hogy abban az esetben, ha üres a forrásunk tartalma, akkor a másolás gombra történő kattintás után nem kerül vissza a fókusz a forrásra, hiszen nem hajtódik végre a másolás kódja. Ezt a legegyszerűbben úgy oldhatjuk meg, ha már az elágazás előtt átadjuk a fókuszt, így mindkét esetben a forrásunk lesz aktív. Persze jó lenne közölni a felhasználóval, ha üresen próbálta átmásolni a forrást a célba, így az igaz ágba is kerülhet egy kiegészítő sor:

Showmessage('A forrás tartalma üres!')

Az utasítás meghívja a showmessage függvényt, mely a paramétereként megadott szöveget kiírja a képernyőre egy párbeszédablak segítségével. Módosítsuk tehát a kódunkat:

```
procedure TfrmMasol.btnMasolClick(Sender: TObject);
begin
edtForras.SetFocus;
  If edtForras.Text='' then
        showmessage('A forrás tartalma üres!')
  Else
        edtCel.text:=edtForras.text;
end;

end.
```

Mentsünk, és futtassunk. Látni fogjuk, hogy a showmessage függvény párbeszédablaka a Caption részében a project nevét tartalmazza, amit nem tudunk megváltoztatni. Természetesen lehetőségünk van saját üzenetablakok szerkesztésére is, ahogy azt majd a későbbiekben látni is fogjuk. Csinosítsuk még egy kicsit tovább a programot:

13. Tervezőnézeten tegyünk fel egy újabb gombot a másolás gomb alá, és állítsuk be a tulajdonságait:
    a. Name: btnTorol
    b. Caption: Törlés
    c. Left: 30
    d. Top: 80

    Kattintsunk duplán a bal gombbal a most beállított törlés gombon, és a megjelenő kódszerkesztőbe írjuk be a törlés utasításait:

    Szeretnénk, ha a forrás mező tartalma törlődne,
    Szeretnénk, ha ez a céllal is megtörténne,

És végül szeretnénk, ha a fókuszt a forrás kapná, hogy a felhasználó újra kezdeményezhesse a másolást:

```
procedure TfrmMasol.btnTorolClick(Sender: TObject);
begin
edtForras.text:=''; //a forrás tartalmának ürítése
edtCel.Text:='';    (* a cél tartalmának ürítése *)
edtForras.SetFocus; { a fókusz forrásra állítása }

end;
```

Kommentezés:

Mint látható, a kódban a kommenteket (magyarázatokat) is elhelyeztük, melyekre a Delphi három lehetőséget is biztosít:

1. A dupla Slash (//) egy sor kommentezését teszi lehetővé, és a komment a következő sor elejéig tart.
2. A (*…*) karaktersorozat közötti rész többsoros kommentet tesz lehetővé, természetesen ekkor a lezáró jelsorozat jelzi a komment végét.
3. A {…} karaktersorozat, mely az előzővel egyenértékű.

Célszerű a kódunkban lehetőség szerint minél több megjegyzést elhelyezni, főleg akkor, ha még csak ismerkedünk a Delphivel. A későbbiek során az alapparancsok kommentezése elhagyható, de egy összetettebb utasítássor előtt mindenképpen jelezzük annak kezdetét, feladatát, és jelentőségét. Egy átlátható, értelmezhető dokumentációnak tökéletes alapja a megjegyzésekkel ellátott kód. Folytassuk most tovább a munkát a Másolás projecttel.

14. Mivel már két gombunk is van a formon, adjunk lehetőséget a felhasználónak arra, hogy egy gomb segítségével is ki tudjon lépni a programunkból. Ezt már az előző alkalommal megtettük a Helló projectnél is, így ismétlésképpen hajtjuk végre a feladatot. Válasszunk ki egy újabb gombot, és helyezzük a Törlés gomb alá. Állítsuk be a tulajdonságait:
    a. Name: btnKilep
    b. Caption: Kilépés
    c. Left: 30
    d. Top: 115

    Kattintsunk a gombon, duplán a bal egérgombbal, és a kódba írjuk be a kilépés utasítását: **close**;

Gyorsbillentyűk előállítása:

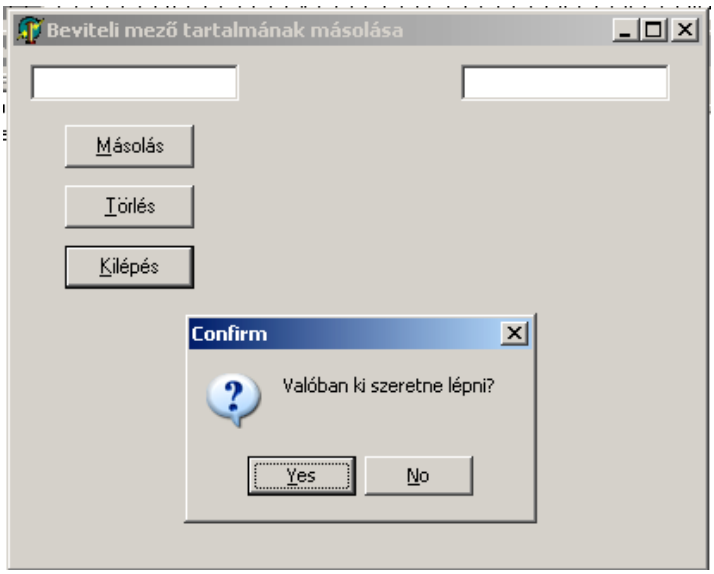
15. Ha szeretnénk biztosítani a felhasználónak a gyorsbillentyűvel való végrehajtás lehetőségét, akkor elegendő, ha a gombok Caption tulajdonságában a kívánt karakter elé betesszük az & jelet. Ennek eredménye, hogy a gombok eseményeit a billentyűzet segítségével is ki tudjuk váltani.
    Válasszuk ki a Kilépés gombot, keressük meg a Caption tulajdonságát, és módosítsuk: &Kilépés.
    Tegyük meg ezt a másik két gombnál is: &Törlés, és &Másolás.
    Ennek köszönhetően most a K, a T és az M billentyűket tudjuk majd használni programunkban. (Ügyeljünk arra, hogy a menüben szereplő gyorsbillentyű kombinációkkal ne ütközzenek a gyorsbillentyűink!)

Kérjünk mentést, majd teszteljük újra programunkat, győződjünk meg róla, hogy a gyorsbillentyűink is működnek (Alt+M, Alt+T, Alt+K). Ha mindent rendben találtunk, egy újabb probléma merül fel: egy alkalmazásból való kilépésnél célszerű meggyőződnünk róla, hogy a felhasználó valóban ki szeretne lépni az alkalmazásból, vagy esetleg véletlenül kattintott a kilépés gombra. A problémát egy, a bezáráskor megjelenő párbeszédablakkal küszöbölhetjük ki, mely megerősítést kér a felhasználótól, és csak abban az esetben engedi bezárni az alkalmazást, ha a válasz igen. Korábban már dolgoztunk egy üzenetablakkal (showmessage), de ez most nem használható, hiszen a felhasználónak nem csupán az OK gombra van szüksége, hanem egy igen-nem választási lehetőségre. Nézzük tehát a megoldást:

16. A megerősítést írhatnánk az általunk létrehozott Kilépés gombra is, de ebben az esetben csupán a gombra történő kattintáskor kérne megerősítést a program, az X mentén történő bezáráskor nem. (Persze elképzelhető, hogy pontosan ez a célunk, akkor oda írhatnánk az eljárásunkat). Mi most mindkét esetben (sőt, még az Alt+F4 billentyűkombinációra is!) megerősítést kérünk.
    Ezt a form eseményei között tudjuk megtenni, kattintsunk tehát az űrlap szürke (üres) területére, vagy az objektum-kezelőben nyissuk le a felső lenyílólistát, és válasszuk ki a formunkat (frmMasol). Lépjünk át az Events (események) fülre, és válasszuk ki az OnCloseQuery eseményt (kattintsunk bele duplán). Az ide írt kód, az ablak bármilyen módon történő bezárásakor végrehajtásra kerül. A Begin…End közé írjuk be a következő kódot:

```
CanClose:=messagedlg('Valóban ki szeretne lépni?',mtconfirmation,[mbYes, mbNo],0)=mrYes;
```

Vizsgáljuk meg a fenti kódot. A CanClose direktíva jelzi a kódban a bezárhatóságot, melyet az üzenetablakra érkező igen válasz ad. Az üzenetablakunk (messagedlg) 4 paraméterrel rendelkezik: az üzenettel (egy általunk megadott szövegkonstans), az üzenetablak típusával (később részletesen beszélünk róla), a megjelenítendő gombok felsorolásával, valamint a súgó környezetazonosítójával. Szó szerinti fordításban a kódunk a következő: „Az alkalmazásablak akkor zárható be, ha a Valóban ki szeretne lépni? kérdésünkre a felhasználó az Igen gombot nyomja meg.”
Mentsünk újra, majd futtassuk az alkalmazásunkat. Próbáljuk ki a kilépést mindhárom lehetséges módon (X, Kilépés gomb, Alt+F4)



Üzenetablakok a Delphiben:

Az előző példafeladatunkban két üzenetablakot is használtunk a felhasználóval való kommunikáció elősegítésének érdekében, nézzük milyen lehetőségeket kínál tehát összességében erre a feladatra a Delphi.

ShowMessage():

Egyszerű üzenetablak, paraméterként csak egy aposztrófok közötti szöveget adhatunk meg. Ezt alkalmazzuk akkor, ha csupán mi üzenünk a felhasználónak, nem választási lehetőséget kínálunk neki.

MessageDlg('Üzenet', ablak típusa, [Gomb1, Gomb2…],súgó)

A Message Dialog (üzenetdialógusablak) már 4 paraméter fogadására képes. Elsőként az üzenet szövegét kell megadnunk szintén aposztrófok között. Vesszővel tagolva következik az ablak típusának meghatározása, ahol csak a 4 szokásos windowsos szimbólummal rendelkező, vagy szimbólum nélküli ablak adható meg:

- mtWarning: Sárga háromszög felkiáltójellel,
- mtError: Vörös körben fehér kereszt,
- mtConfirmation: Buborékablakban egy kérdőjel,
- mtInformation: Buborékablakban egy I betű,
- mtCustom: Grafika nélküli üzenet.

Vesszővel tagolva következnek a gombok szögletes zárójelek között: [mbYes, mbNo], ahol az mb (MessageButton), a név pedig az angol megfelelő. A szögletes zárójel arra szolgál, hogy belül felsorolhassuk (szintén vesszővel tagolva) a megjelenítendő gombokat. Az utolsó paraméter, a súgó környezetazonosító megadása lenne, ha ezt nem szeretnénk használni, akkor is meg kell adnunk egy egész értéket: 0.

Ebben az üzenetablakban sajnos még mindig nincsen lehetőségünk, az ablak fejlécében szereplő feliratot (Caption) megváltoztatni, bár láthatóan tökéletesen alkalmas a felhasználóval való kommunikáció lebonyolítására.

MessageBox('Üzenet', 'Üzenet fejléce', választható gombok)

A MessageBox (üzenetdoboz) 3 paraméterrel rendelkezik: az üzenettel, az üzenet fejrészének megadásának lehetőségével, és a gombok felsorolásával. Ennek segítségével már lehetőségünk van saját fejléccel rendelkező üzenetek megadására. A MessageBox, és a MessageDlg azonban függvényként működik, így a létrehozásukkor függvényként tudjuk csak őket vizsgálni (vagyis elágazásba ágyazva nézhetjük meg értékeiket.)

Térjünk vissza a korábbi feladatunkhoz, és egészítsük ki a kódunkat a most megismert üzenetablakkal, mely csak akkor engedi a másolást, ha a felhasználó tényleg másolni szeretne.

17. Tervezőnézetben kattintsunk duplát a Másolás gombunkon, hogy előrehozzuk a kódszerkesztőt. A kódunkban már korábban elhelyeztünk egy elágazást, mely a beviteli mező üres állapotát vizsgálva nem hajtja végre a másolást. Most mélyítsük tovább az ágakat. Ha megerősítést szeretnénk kérni, azt nyilván a másolás megkezdése után kell csak megtennünk, tehát oda kerül a kód. Kattintsunk az Else utasítás után, üssünk egy Entert, és írjuk be a kódot:

If Application.MessageBox('Másolhatok?','Megerősítés!',mb_YesNo)=IDYes then

A kód magyarázata: Ha az üzenetablak két választható gombja közül (Igen, Nem) a felhasználó az Igen-t (pontosabban annak azonosítóját) választja, akkor…
És itt folytatódik a kódunk a tényleges másolásutasítással.

```
procedure TfrmMasol.btnMasolClick(Sender: TObject);
begin
edtForras.SetFocus;
  If edtForras.Text='' then
        showmessage('A forrás tartalma üres!')
  Else
        If Application.MessageBox('Másolhatok?','Megerősítés!',mb_YesNo)=IDYes then
        edtCel.text:=edtForras.text;
end;
```

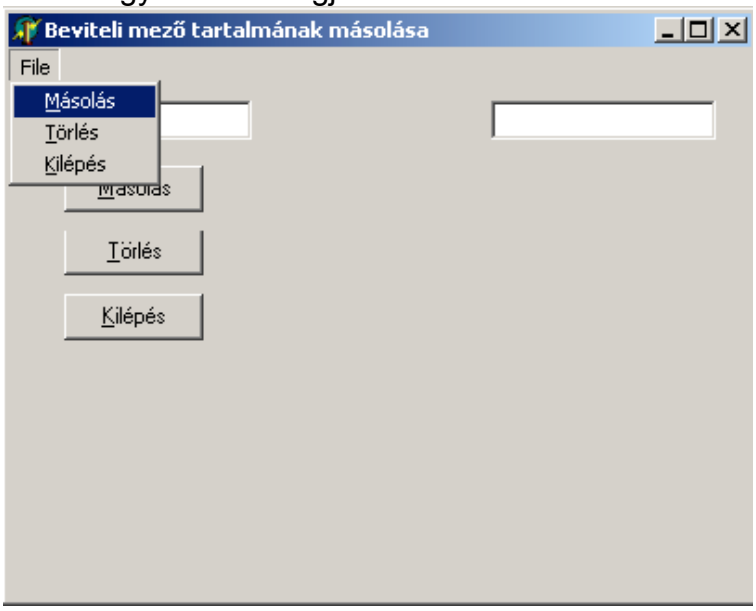
Bár némiképp eltérő a MessageBox, és a MessageDlg, mégis sok közös vonásuk van. A kezdeti időkben érdemes kommentezni a párbeszédablakainkat, amíg hozzászokunk, hogy mely esetben melyik változatot kell használnunk. Ha mentünk, és futtatunk, nézzük meg, hogy a MessageDlg-vel ellentétben a MessageBox gombjai magyarul „beszélnek”.
Van még egy apróság: a MessageDlg esetén van alapértelmezett (kijelölt) gombunk (az első Yes), míg a MessageBox-nál nincsen. Pedig ha kipróbáljuk, akkor tapasztalni fogjuk, hogy mégis működik az alapértelmezett gomb: Yes, csak nincsen oly mértékben kiemelve, mint a másik ablak esetén (nem kapja meg közvetlenül láthatóan a fókuszt).
Itt is érdemes segítségül hívnunk a súgót ha elakadnánk: jelöljük ki a kódban a kívánt szövegrészt (MessageBox), majd nyomjuk le az F1-et.

Menükészítés a Delphiben:

Tulajdonképpen készen is vagyunk, de használjunk fel még egy elemet, ha már a standard komponenspalettával foglalkoztunk. Készítsünk el egy menüt az alkalmazásunkhoz.

18. Keressük meg a komponenspalettán a MainMenu elemet, kapcsoljuk be, majd helyezzük el a formunkon. Mivel nem fog látszani futásidőben, így mindegy hová tesszük. Állítsuk be a név tulajdonságot: Name: mnuMasol.
    Kattintsunk duplán a formon lévő most elhelyezett MainMenu elemre. A megjelenő párbeszédablakban már csak a menüelemeket kell felvennünk:
    Mivel az első menühely aktív, írhatjuk is az első menücímet: "&File" (emlékezzünk: az & jel a gyorsbillentyű funkcióért felelős). Vegyük észre, hogy a szöveget tulajdonképpen az Object Inspector Caption cellájába írjuk be. Az Enter megnyomása után megnyílik a File menü alá tartozó menütételek első eleme. Ide írjuk be a "&Másolás"-t, a "&Törlés"-t, majd a "&Kilépés"-t. Ezzel végeztünk is a menü elkészítésével, már működne, de még nem rendeltünk

hozzá eseményeket, tehát egy mentés, és próba után ezzel folytatjuk. A futtatás során most egyelőre ezt fogjuk látni:



Korábban megírt eljárások hívása:

Ne essünk kétségbe, ha nem látszanak az aláhúzott gyorsbillentyűk, a funkció aktiválásakor (Alt+F) láthatóak lesznek az elemeink. Tegyük most tehát ezeket működőképessé. Akár azt is tehetnénk, hogy a korábban megírt eljárásainkat újra megírjuk a megfelelő menüpontokhoz (ekkor arról teszünk tanúbizonyságot, hogy még a windows alapjait sem ismerjük, lásd: szerkesztés, másolás, beillesztés). Ha viszont ez utóbbi módszert választjuk (kimásoljuk a kódot, és beillesztjük), akkor pedig arról győzzük meg majd a kódba betekintő szemlélőt, hogy az objektumorientált programozással kapcsolatban (finoman fogalmazva) felszínesek az ismereteink. Ha észrevettük, valamennyi eddig írt kódunk procedure, vagyis eljárás, tehát a Unitunkon belül bármikor, bárhonnan meghívható, végrehajtható. Nézzük hogyan:

19. A formunk tervezőnézetében kattintsunk duplán a MainMenu elemünkre. A megjelenő (korábban már használt) menüszerkesztő párbeszédpanelen kattintsunk szintén duplán az első menüelemünkre: a Másolásra. Ennek eredményeképpen a kódszerkesztőben azonnal megjelenik a formunkon lévő menü objektumunk másolás elemének kattintásra történő eseménye:

```
procedure TfrmMasol.Msols1Click(Sender: TObject);
begin

end;

end.
```

Látható, hogy mivel nem adtunk nevet a menüelemünknek, így a Delphi konvenciók érvényesek, vagyis a Másolás menünek a Delphi adta a Msols1 nevet (az általunk adott név ékezetes karakterek nélküli változata+azonosító). Természetesen ezt, ha szeretnénk, átnevezhetnénk az objektum-kezelőben, de számunkra most ez

mellékes, így nem foglalkozunk vele. A Begin…End közé be kellene írnunk annak az eljárásnak a meghívásához tartozó kódot, melyet a Másolás gombnál megírtunk:

20. A kódszerkesztő ablakában a görgetősáv segítségével (felfelé indulva) keressük meg a szükséges kódot (Látni fogjuk, hogy az általunk keresett kód a: **procedure TfrmMasol.btnMasolClick(Sender: TObject);**). Innen bennünket az eljárás neve érdekel: **TfrmMasol.btnMasolClick**, melyből természetesen az osztályazonosító (T) jelet majd el kell hagynunk. Jegyezzük ezt meg, majd görgessünk vissza a menüelemünk kódjához. Már tudjuk, hogy mit kell meghívnunk, lássuk, hogyan hívhatjuk meg. Írjuk be a következő utasítást a menüelemünk Begin…End-je közé:

    FrmMasol.btnMasolClick(Sender);

    Eljárást, függvényt paraméterek nélkül nem igazán tudunk meghívni, ezért kerül a kód végére zárójelek közé a fenti paraméter. Ha mentünk, és kipróbáljuk a kódunkat, látni fogjuk, hogy minden rendben, működőképes a menünk is. Egészítsük ki a kódunkat a maradék két menüponthoz tartozó utasítással, mentsünk, és futtassunk.

```
procedure TfrmMasol.Msols1Click(Sender: TObject);
begin
frmMasol.btnMasolClick(Sender);
end;

procedure TfrmMasol.Trls1Click(Sender: TObject);
begin
frmMasol.btnTorolClick(Sender);
end;

procedure TfrmMasol.Kilps1Click(Sender: TObject);
begin
close;
end;

end.
```

Láthatjuk, a kilépésnél nem érdemes bonyolítani az életünket, elegendő, ha a rövidebb, célratörő kódot írjuk be, hisz a hozzátartozó megerősítések a form OnCloseQuery eseményénél vannak definiálva. Mentsünk, és dőljünk kényelmesen hátra, ma hasznos elemeket sikerült megismernünk a Delphi komponenseiből, melyekkel már látványos alkalmazásokat készíthetünk. Az eredeti Másolás projectünk egy kicsit összetettebb lett a korábbi terveinkhez képest (sőt, túlságosan is összetett!), de a tanulmányaink elmélyítéséhez nagymértékben segítségünkre volt.

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Mire használhatjuk a Delphiben a Standard komponenspalettát?
   a., Kódok egyszerűbbé tételére
   b., Vizuális elemek elhelyezésére a formon
   c., Komponensek kijelölésére
2. Melyik nem eleme a standard komponenspalettának?
   a., PopupMenu
   b., MainMenu
   c., StatusBar
3. Melyik elem nem jelenik meg futásidőben a formon?
   a., ScrollBar
   b., ActionList
   c., Panel
4. Melyik utasítással adható át a fókusz egy vezérlőelemre?
   a., SetFocus
   b., AddFocus
   c., FocusAdd
5. Melyik üzenetablak alkalmas csak információ megjelenítésre?
   a., MessageBox
   b., ShowMessage
   c., MessageDlg
6. Hány lehetőséget kínál a delphi megjegyzések készítésére?
   a., 3
   b., 5
   c., 7
7. A következő típusok közül melyiket használjuk a megerősítés üzenetablak (buborékablakban egy kérdőjel) ikonjának megjelenítésére a MessageDlg függvényben?
   a., mtInformation
   b., mtCustom
   c., mtConfirmation

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. A Delphiben 5 lehetőségünk van kommentek készítésére.
   igaz
   hamis
2. A MessageDlg fejrésze (Caption) paraméterezhető.
   igaz
   hamis
3. A SetFocus utasítással átadható a fókusz egy vezérlőelemre.
   igaz
   hamis
4. Az egyik szabványos kommentezési lehetőség a Delphiben: \\
   igaz
   hamis
5. A „quit” utasítással kezdeményezhető a form bezárásának művelete.
   igaz
   hamis
6. A MessageBox függvényben megadható az üzenetablak fejrésze is.
   igaz
   hamis
7. A standard komponenspalettáról lenyíló listaelemeket is választhatunk.
   igaz
   hamis
8. A Delphiben lehetőségünk van egyéni komponensek definiálására is
   igaz
   hamis

III.

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Sorolja fel a Delphi kommentezéseinek lehetőségeit!
2. Sorolja fel a Delphi üzenetek megjelenítésére szolgáló elemeit!
3. Elemezze részletesen a Delphi MessageDlg függvényét!
4. Vázolja fel a menükészítés lehetőségét a Delphiben!
5. Mi a CanClose direktíva, és hol, mire használjuk?
6. Mire alkalmas a ShowMessage?

A Delphi komponensei II.

Az előző fejezetben megismerkedtünk a standard komponenspaletta elemeivel, folytassuk most tovább a következő komponenspalettával.

Additional komponenspaletta:

Itt a ritkábban használt komponensek találhatóak meg, melyeket az általános alkalmazásoknál használni szoktak. Mint látható, a bal szélen az első komponens ismét a Nyíl, mely a kijelölés biztosításának lehetőségéért található fent. A többi:

- BitBtn: Szöveggel, és ábrával ellátható nyomógomb.
- SpeedButton: Grafikával ellátható nyomógomb.
- MaskEdit: Maszkolható bevitelimező.
- StringGrid: Szövegtáblázat.
- DrawGrid: Képek táblázatos megjelenítése.
- Image: Képek megjelenítése.
- Shape: Grafikai alakzatok megjelenítése.
- Bevel: 3D vonalak, és négyszögek.
- ScrollBox: Görgethető felület.
- CheckListBox: Választható lista.
- Splitter: Tagoló, területfelosztó.
- StaticText: Statikus címke.
- ControlBar: Eltávolítható eszköztár létrehozására szolgál.
- ApplicationEvents: Alkalmazásunk eseményeinek figyelésére.
- Chart: Egy táblázat adatainak diagram formátumú megjelenítésére szolgál.

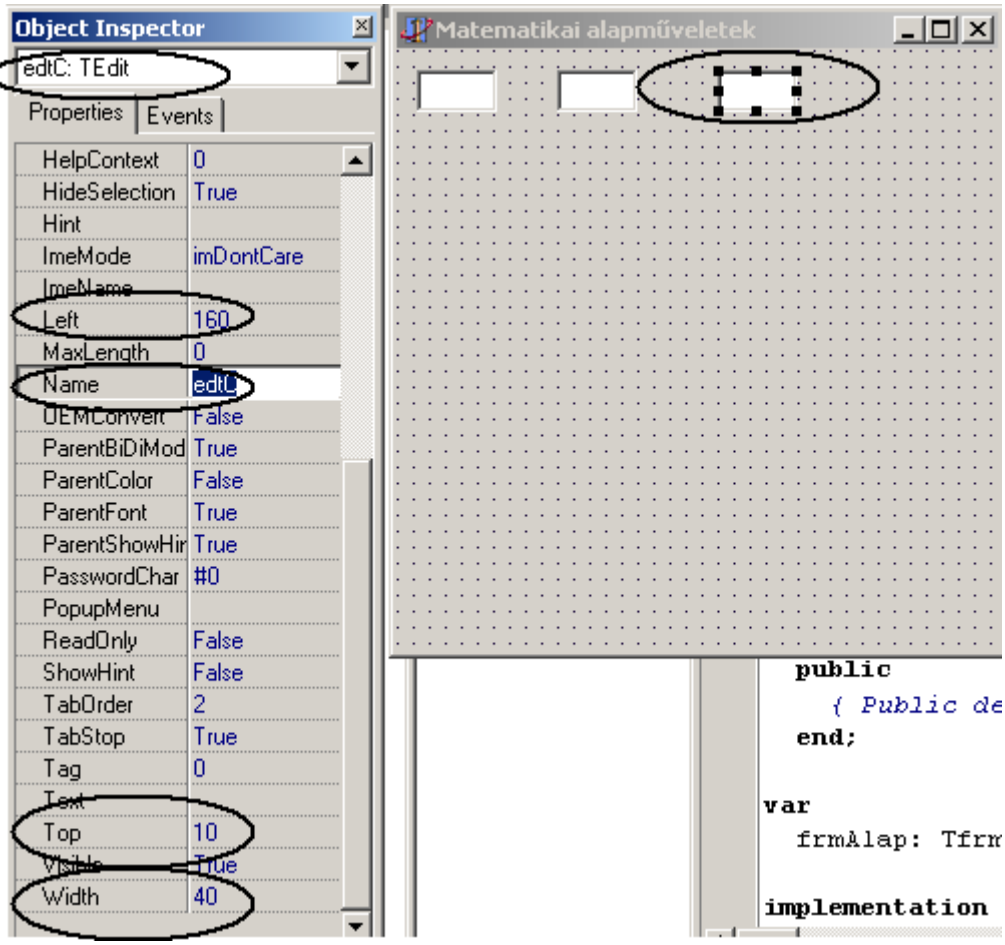
A fenti elemek közül a BitBtn, és a SpeedButton emelkedik ki különösen, ugyanis ezek segítségével készíthetünk eszköztárat (a Standard Button ugyanis nem ruházható fel kép tulajdonsággal). Különleges jelentősége van még a MaskEdit-nek is, melynek akkor vesszük hasznát, ha egy beviteli mező feltöltésekor korlátozni szeretnénk a bevihető adatokat. Ez megkerülhető lenne a kódban történő ellenőrzéssel (hibakezeléssel), de a feladat gyakorisága miatt a fejlesztők elkészítették ezt a komponenst is. Az Image alapesetekben a formok díszítésére használható (képet tudunk benne megjeleníteni), de mint később majd látni fogjuk az alkalmazásainkba is látványos elemként elhelyezhető.

Feladat: Matematikai alapműveletek (4Alap)

A mai alkalommal elkészítünk egy egyszerű alkalmazást, mely matematikai alapműveletek végrehajtását teszi lehetővé. Az Intézőben hozzuk létre a Delphi almappán belül a 4Alap nevű alkönyvtárat (ezzel a névvel utalunk a 4 matematikai alapműveletre). És kezdhetjük is a munkát:

1. Indítsuk el a Delphit, majd a megjelenő ablakban a form tulajdonságait állítsuk be a következők szerint:
   a. Name: frmAlap
   b. Caption: Matematikai alapműveletek
   c. ClientWidth: 250
   d. ClientHeight: 230
2. Kérjünk mentést a Save All-al, a Unit neve Ualap, a project neve Palap, melyek kerüljenek az előzőleg létrehozott 4Alap könyvtárba.
3. Hozzuk létre az adatok feldolgozásához szükséges beviteli mezőket a következők szerint. Válasszuk ki a Standard paletta Edit elemét, és helyezzük a formon a bal felső sarokba. A tulajdonságait módosítsuk a következők szerint:
   a. Name: edtA
   b. Text: töröljük ki a tartalmát
   c. Left: 10
   d. Top: 10
   e. Width: 40

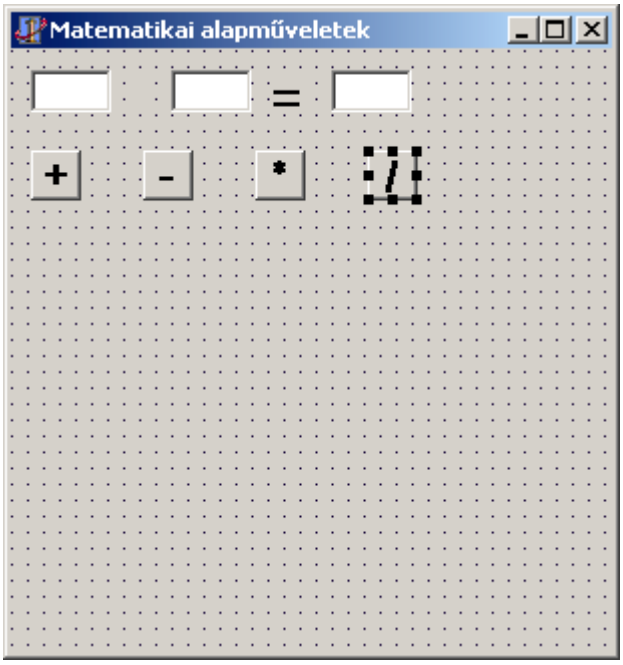
   Ebből az elemből szükségünk lesz még kettőre (edtB, és edtC), melyeket helyezzünk vízszintesen az első beviteli mező mellé (edtB Left tulajdonsága: 80, edtC Left tulajdonsága: 160), hogy a következő képet lássuk:



4. Készítsünk most el két címkét, egyik a műveleti jel megjelenítéséért felelős, a másik pedig az egyenlőségjelet rajzolja ki. Válasszuk ki a Label komponenset, majd a formon az első két beviteli mező közé kattintsunk egyet. A megjelent címkénk tulajdonságait módosítsuk a következő módon:
   a. Name: lblMuv
   b. Caption: töröljük a tartalmát (majd a műveletnek megfelelően alakul)
   c. Font: Size: 18, félkövér (a Font mögötti … ikon-ra kattintsunk)
   d. Height: 21 (ugyanakkora, mint az Edit alapértéke)
   e. Left: 60 (nagyjából középen)
   f. Top: 10 (mint az Edit-nél)

   Jöhet a következő Label komponens, amit a második, és harmadik Edit közé helyezünk el. Tulajdonságai:
   a. Name: lblEred
   b. Caption: = (vagyis egy egyenlőségjel)
   c. Font: mint az előző címkénél
   d. Height: 21
   e. Left: 130 (elég nagyjából, majd módosíthatjuk)
   f. Top: 10
5. Készítsük most el az alapműveletek végrehajtásáért felelős gombokat. Válasszuk ki a Button komponenst, majd helyezzük el a formra a beviteli mezők alá bal szélre. Tulajdonságai:
   a. Name: btnOsszead
   b. Caption: + (vagyis egy összeadásjel)
   c. Height: 25
   d. Left: 10
   e. Top: 50
   f. Width: 25

   Helyezzünk el még 3 gombot: kivonás (Name btnKivon, Caption: -), szorzás (Name: btnSzoroz, Caption: *), osztás (Name: btnOszt, Caption: /) az előző mellé, hogy a következő képet lássuk:



Adatok átalakítása, konvertálása Delphiben:

A számítási műveletek végrehajtása előtt néhány dolgot végig kell gondolnunk. Ha feltételezzük, hogy a felhasználók tudják, mit kell tenniük, akkor koncentrálhatunk először a lényegi feladatokra, vagyis ha az összeadásra kattintunk, szeretnénk a felhasználó által az első, és második mezőbe bevitt értékeket összegezni, és megjeleníteni a harmadik mezőben, miközben a két első mező közt elhelyezkedő címke felveszi az összeadás szimbólumjelét. Első nekifutásra nem tűnik túl nehéznek a feladat, de rögtön adódik egy apró probléma: a beviteli mezők szöveg típusú adatokkal dolgoznak, nekünk viszont össze kellene adnunk két számot. Hogy értsük miről van szó, az elején kezdjük:

6. Tervezés alatt kattintsunk duplán az összeadás elvégzéséért felelős gombunkra. A kódszerkesztőt fogjuk megkapni, próbáljuk elvégezni az összeadást. Első ötletünk lehet a következő kód:

```
procedure TfrmAlap.btnOsszeadClick(Sender: TObject);
begin
edtC.text:=edtA.Text+edtB.text;
end;

end.
```

Ezzel a kóddal azt fogjuk elérni, hogy a két beviteli mező tartalmát a Delphi összefűzi, vagyis (pl) 5+5=10 helyett az eredmény: 5+5=55. A gond esetünkben az, hogy a beviteli mező alapértelmezetten szöveges adatokat tárol, mi viszont most számítást végeznénk. Mi tehát a teendő? Át kell alakítanunk a szövegeket számmá, majd így elvégezni a műveletet. Természetesen az átalakításhoz változókra lesz szükségünk, így a Begin utasítás előtt (a Pascal szabályoknak megfelelően) definiálnunk kell a kívánt változókat. Ne felejtsük el, hogy míg a művelethez átalakításra van szükség, addig a művelet végén vissza is kell tudnunk alakítani a számot szöveggé. Három változóra lesz tehát szükségünk: a,b,c:integer típusokra. Definiáljuk őket:

```
procedure TfrmAlap.btnOsszeadClick(Sender: TObject);
var a,b,c:integer;
begin
edtC.text:=edtA.text+edtB.Text
end;

end.
```

Majd módosítsuk a kódot. Tudjuk, hogy először a mezők tartalmát számmá kell alakítanunk, ehhez a Delphi StrToInt (StringToInteger, SzövegbőlEgész) függvényét hívjuk segítségül, majd elvégezzük a matematikai műveletet, és visszaalakítjuk az eredményt szöveggé, az IntToStr függvény segítségével:

```
procedure TfrmAlap.btnOsszeadClick(Sender: TObject);
var a,b,c:integer;
begin
a:=StrToInt(edtA.Text);
b:=StrToInt(edtB.Text);
c:=a+b;
edtC.Text:=IntToStr(c);
end;

end.
```

Adjuk még ehhez hozzá a címkét megrajzoló: **lblMuv.Caption:='+';** utasítást, és mentés után teszteljük az eredményt.

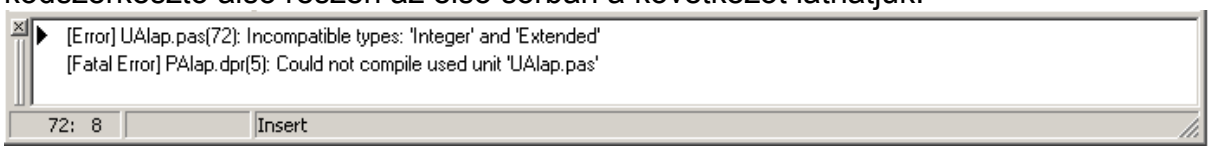
A következő lépés a kivonás, melyben gyakorlatilag minden megegyezik a műveleti jelek kivételével (átalakítunk, elvégezzük a műveletet, és visszaalakítunk)

```
procedure TfrmAlap.btnKivonClick(Sender: TObject);
var a,b,c:integer;
begin
a:=StrToInt(edtA.Text);
b:=StrToInt(edtB.Text);
c:=a-b;
edtC.Text:=IntToStr(c);
lblMuv.Caption:='-';
end;

end.
```

Természetesen mindezt a kivonás gomb kattintásra történő eseményéhez írtuk.

Fordítás közbeni hibaüzenetek:

Másoljuk ki a kódot, és illesszük be a szorzás, és az osztás helyére is, módosítsuk a szükséges részeket, majd ellenőrizzük a programunkat, hogy minden rendben történik-e? Amikor futtatni akarunk, a Delphi hibaüzenetet fog küldeni: a kódszerkesztő alsó részén az első sorban a következőt láthatjuk:



A hiba az integer típussal van, mégpedig az osztás során (a kurzor az osztás műveletén áll : c:=a/b; mi lehet a baj? Természetesen nem az A, és B változókkal van a gond (mert akkor ennek már korábban jelentkeznie kellett volna), marad a C változó, és valóban. A Delphi a hibaüzenettel hívja fel figyelmünket arra, hogy két egész típusú változó (a, b) osztása esetén egyáltalán nem garantálható hogy egész típus lesz az eredmény! (sőt) Mi most a teendő? Módosítsuk a c változó típusát helyileg, vegyük ki az integer típusok közül, és definiáljuk double típusúként. Erre (bár nem ezt várnánk) megint hibaüzenetet kapunk, de most az eredmény kiíratásakor. Ha figyelmesen megnézzük a hibaüzenetet, látni fogjuk mi a gond: a kiíráskor az IntToStr függvényt használtuk, de az előbb megváltoztattuk a c típusát, így az már nem Int. hanem double, amit a FloatToStr (Lebegőpontosból Szöveggé) függvény tud átalakítani. Javítsuk tehát ki a kódot, mentsünk, és reménykedjünk, hogy nem fog elszabadulni a pokol…

```
procedure TfrmAlap.btnOsztClick(Sender: TObject);
var     a,b:integer;
        c:double;
begin
a:=StrToInt(edtA.Text);
b:=StrToInt(edtB.Text);
c:=a/b;
edtC.Text:=FloatToStr(c);
lblMuv.Caption:='/';
end;

end.
```

És végre minden rendben, a kódunk megfelelően működik, programunk összead, kivon, oszt, szoroz. Sajnos tudomásul kell vennünk, hogy minden programfejlesztő saját maga alatt vágja a fát, ugyanis a feladat megfogalmazásakor már a hibakezelések tömegét fogjuk magunkra szabadítani.

Hibakezelések a delphiben:

Korábbi modulokon belül már volt szó a programok hibáiról, melyek alapvetően két csoportra bonthatóak:

Runtime (futásidejű) hibák:

Ezek ellen kevésbé tudunk védekezni. Legtöbbször a programozón kívülálló okok vezetnek ezekhez (floppyra történő mentés során nincs lemez a meghajtóban, vagy a nyomtató nincsen bekapcsolva, esetleg a hardverek haldokolnak). Az ilyen jellegű hibák megszakítják a programunk szabályszerű futását, és az operációsrendszer megfelelő hibakezelő rutinja veszi át a vezérlést. Természetesen igyekeznünk kell ezek ellen is védekezni, hisz ha elveszítjük az irányítást, nem biztos, hogy visszakapjuk majd a továbbfutás lehetőségét.

Programozói hibák:

Amik a korábban már többször említett két fő csoportba sorolhatók: szintaktikai, és szemantikai hibák. A kezdő programozók leggyakrabban az első csoportba sorolt hibákat követik el (elgépelések, azonosítók helytelen megadása, deklarációs problémák), míg a későbbiekben egyre gyakrabbakká válnak a szemantikai hibák (a program logikai hibái, tömbön túli indexelések, elágazások, ciklusok helytelen felépítései).

Ha szeretnénk, hogy az irányítás végig a kezünkben maradjon, törekednünk kell mindkét fenti hibacsoport előforduló hibáit magunknak kezelni, megakadályozva ezzel, hogy programunk felett az operációsrendszer vegye át az uralmat. Erre a delphi is kínál lehetőségeket:

A RAISE utasítás:

A hibakezelés lényege, hogy a fejlesztés során felfedezve a hibát egy raise utasítást helyezünk el a kódban, melyben megvizsgálhatjuk, hogy bekövetkezik-e a hiba, és amennyiben igen, úgy mi kezelhetjük azt, mielőtt az irányítás kikerülne a kezünkből. (konkrét példát is találunk a delphi súgójában ha a raise utasításra rákeresünk).

A TRY…EXCEPT…END szerkezet

Maga a szerkezet hasonlít az elágazások felépítésére. A TRY utasítás után adjuk meg, milyen művelete(ke)t kell végrehajtani, majd az EXCEPT utasítást követően adjuk meg a kivétel(ek) kezelését. Itt a kivételek kezelése nem más, mint az, hogy hajtsa végre a program az utasításokat, kivéve ha…, mert akkor… A szerkezetre szintén találunk példát a súgóban.

A TRY…FINALLY…END szerkezet:

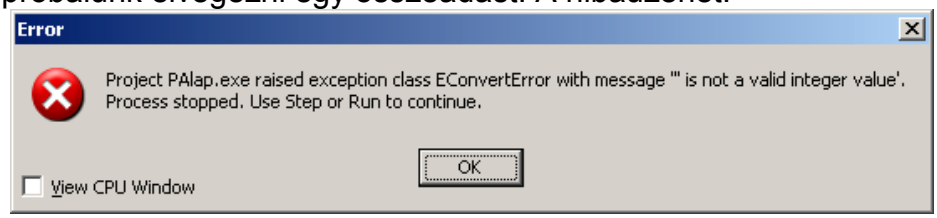
A TRY FINALLY szerkezetet akkor használjuk, ha egy összetett utasítássorozat minden elemének végre kell hajtódnia, de előfordulhat, hogy ez mégsem sikerül. Ekkor az első részben a végrehajtandó utasítások szerepelnek majd, és a FINALLY tartalmazza majd a sikeres műveletek végrehajtása utáni zárással kapcsolatos tennivalókat.

Tesztelés, hibakeresés:

A fenti hibakezelések bármelyikét használhatjuk a programok fejlesztése során, de nem szabad elfeledkeznünk arról, hogy hibát csak úgy találhatunk, ha körültekintően keressük azt! A tesztelésnek nem az a célja, hogy meggyőzzük magunkat a programunk hibátlan voltáról (tudomásul kell végre vennünk, hogy: „nincs tökéletes program, csak olyan, melynek még nem találták meg a hibáit"), hanem az, hogy minél több hibát ki tudjunk javítani a fejlesztés alatt, mielőtt kikerül a kezünkből a program.

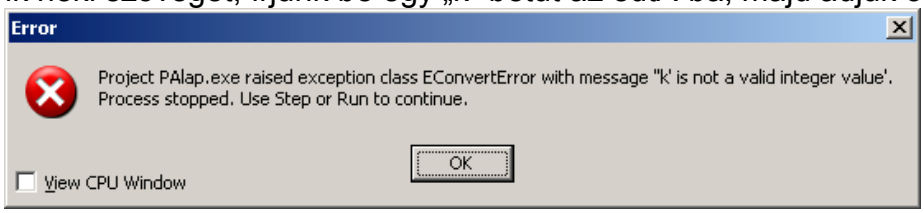
Nézzük rögtön az alapműveleteket végző alkalmazásunkat. Mint megállapítottuk, tökéletesen működik, számol, ahogy azt a gomboktól várjuk. Az elkövetkezendőkben szándékosan hibákat fogunk elkövetni programunkban, mely a Delphi futásának azonnali hibaüzenettel kísért megszakítását fogja eredményezni. Ez önmagában még nem lenne probléma, de ne feledkezzünk el róla, hogy a valós idejű fejlesztés esetén egy hiba a valós idejű program futását szakítja meg. Ha hibaüzenetet kapunk, igyekezzünk értelmezni azt, majd a nyugalmunk megőrzése érdekében a háttérben futó Delphi Run menüpontját választva szakítsuk meg a valós idejű futtatást a Program Reset paranccsal, és teszteljünk tovább… Nézzük hát most egy picit tüzetesebben meg a programunkat:

- Kezdjük mondjuk azzal, hogy üresen hagyjuk a beviteli mezőket, és megpróbálunk elvégezni egy összeadást. A hibaüzenet:



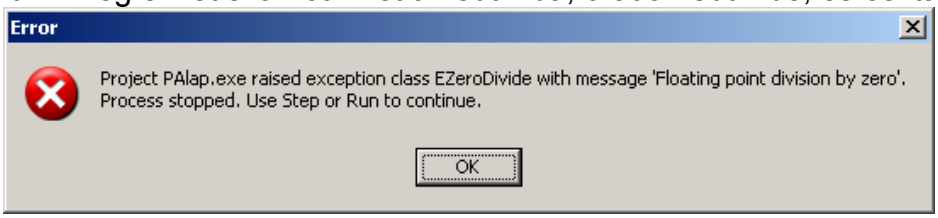
Vegyük tudomásul az üzenetet, majd a fent említett Run/Program Reset parancsot hajtsuk végre. Természetesen eszünkbe nem jutna úgy futtatni a programot, hogy nem adunk meg értékeket, de ezt a Mariska néni nem tudja… Ő csak kattint, ami ezt a hibát adja majd. A hiba oka: át szeretnénk alakítani egy szöveget számmá… de nincs szöveg… Pech, próbáljunk mást:

- Adjunk neki szöveget, írjunk be egy „k" betűt az edtA-ba, majd adjuk össze…



Bizony, a „k" tényleg nem valós egész érték… Sajnos ugyanezt üzeni majd a hibaüzenet, ha nem egész számokat szeretnénk összegezni. Talán érezhető, hogy a programunk elég szerény tudással rendelkezik (egy tesztelő nem 5+5=10-el tesztel…)

- Osszunk még el két számot: 4-et az edtA-ba, 0-át az edtB-be, és osztás…



És nem folytatjuk… A hibaüzenetekkel lassan a fejlesztések során szépen meg fogunk barátkozni, csak a hibaüzenet fontos részeire kell koncentrálnunk, és a háttérben futó kódnak kell előttünk lennie, hogy ne tudjon kihozni bennünket saját programunk a sodrunkból.

Hibajavítások az alkalmazásunkban:

Próbáljuk meg korrigálni a fenti hibáinkat, hogy legalább ez a három hibaüzenet ne jelenjen meg a programunk futása során:

7. Az első hibát egyszerűen korrigálhatjuk, ha a kódban egy elágazásba helyezzük az utasításainkat, melyben először megvizsgáljuk a két alapadat mezőjének ürességét, és csak akkor hajtjuk végre a műveletet, ha azok tartalmaznak értékeket. Természetesen stílusosan külön érdemes megvizsgálnunk az ürességet, így a valóban üres mezőre tudjuk visszaadni a felhasználónak a fókuszt:

```
procedure TfrmAlap.btnOsszeadClick(Sender: TObject);
var a,b,c:integer;
begin
If edtA.Text='' then begin
        showmessage('Adja meg az első elem értékét!');
        edtA.SetFocus;
        end
Else begin
     If edtB.Text='' then begin
                showmessage('Adja meg a második elem értékét!');
                edtB.SetFocus;
                end
     else  begin
        a:=StrToInt(edtA.Text);
        b:=StrToInt(edtB.Text);
        c:=a+b;
        edtC.Text:=IntToStr(c);
        lblMuv.Caption:='+';
        end;
     end;
end;
```

Természetesen ezt a műveletet a többi eseménynél is végre kell hajtanunk, vagyis oda is bekerül a fenti elágazás. Felmerül a kérdés: ha mind a négy esetben megvizsgáljuk a beviteli mezőket, akkor nem lenne-e egyszerűbb ezt egyetlen procedure-ben végrehajtani? Tovább is vihetjük a felmerült gondolatot: miután az átalakításokat is minden esetben végrehajtjuk, nem lenne célszerű ezt is abban a procedure-ben elhelyezne? Dehogynem. A programfejlesztés utolsó fázisáról van most szó, amikor az elkészült működő(!) programunkat megpróbáljuk ésszerűsíteni, hatékonyabbá tenni. Ezt a feladatot Önökre bízzuk, azaz győződjenek meg róla, milyen módon tehető átláthatóbbá, hatékonyabbá, és mindeközben stabilabbá, és gyorsabbá a program? Minden törekvés, mely egy kód ésszerűsítését (rövidítését) célozza megfizethetetlen tanulságokkal, és tapasztalatokkal jár. De nézzük a másik problémát:

8. A nullával való osztást is könnyen kiküszöbölhetjük, hisz, ez csupán egy újabb elágazás, melyet csak az osztás esetén kell vizsgálnunk:

```
procedure TfrmAlap.btnOsztClick(Sender: TObject);
var     a,b:integer;
        c:double;
begin
If edtA.Text='' then begin
        showmessage('Adja meg az első elem értékét!');
        edtA.SetFocus;
        end
Else begin
     If edtB.Text='' then begin
                showmessage('Adja meg a második elem értékét!');
                edtB.SetFocus;
                end
     else  begin
        a:=StrToInt(edtA.Text);
        b:=StrToInt(edtB.Text);
        if b=0 then begin
                showmessage('Nullával nem osztunk!');
                edtB.setfocus;
                end
        Else begin
        c:=a/b;
        edtC.Text:=FloatToStr(c);
        lblMuv.Caption:='/';
        end;
        end;
     end;
end;
```

Látható, hogy a kódok írásakor nem kommenteztük a szöveget, alapozunk ugyanis a meglévő Pascal ismereteikre, ahol az elágazásokkal kapcsolatos alapokat már elsajátítottak. Az Object Pascal ebből a szempontból teljesen egyenértékű, így gond nélkül használhatják fel ismereteiket.

9. Az utolsó probléma a mezők tartalmából adódott (ha emlékszünk még a „k” betűre, vagy a nem egész értékekre), s bár elsőre nem tűnik túl komolynak a hiba, mégis ezzel lehet a legtöbb időt tölteni.

Kezdjük az egyszerűbb falattal: dolgozzuk át a programot, hogy ne csupán egész értékeket legyen képes elfogadni. Ezt részben már megtettük, amikor az osztás eredményeként kaphattunk nem egész (double) típusú adatot. Nincs más dolgunk, mint átdolgozni a kódot: az a, b értékét double-ként definiálni (mivel külön írtuk a kódot, így minden eljárásban), az átalakításokat StrToInt helyett StrToFloat-ra módosítani, valamint a visszatérést FloatToStr-re…

Ha végeztünk, próbáljuk ki a nem egész értékekkel történő műveleteket. Ha azt gondoljuk minden rendben, akkor a tizedesvessző helyett (tudják: Mariska néni), használjuk a pontot… vagy a korábban már próbált „k” betűt…

A gond természetesen a karakterekkel van, ezt egyértelműen láthattuk most is. Mi lehet a megoldás? Csak számot szabad engednünk a beviteli mezőbe írni.

Ha megnézzük az Edit mezőnket, tapasztalni fogjuk, hogy nem tudjuk korlátozni az általunk meghatározott módon a bevitelt, tehát barkácsolnunk kell. Sok lehetséges (elég összetett, és bonyolult) megoldás létezik, de itt az ok, hogy miért hozták létre a MaskEdit elemet a delphi fejlesztői…

Átalakítások, a VAL függvény:

Mi most egy másik úton fogunk elindulni. A rendszerfüggvények között megtalálható a Val függvény, melyet átalakításokra használhatunk. Ennek előnye, hogy amennyiben nem számot ad meg a felhasználó, a kódunk akkor is hiba nélkül fog lefutni. Természetesen az eredmény az értelmezhető adatokból kerül majd ki. További előnye a VAL függvénynek, hogy kikerüli az értelmezhetetlen adatokat, így nem kell megnéznünk, Mariska néni kitöltött-e minden mezőt. Ez persze a kódunk egyszerűsítésével, gyakorlatilag annak átírásával is jár. Módosítsuk a feladatunk összeadásának kódját:

```
procedure TfrmAlap.btnOsszeadClick(Sender: TObject);
var a,b,c:double;
    q,e:integer;
begin
  lblMuv.Caption:='+';
  val(edtA.text,a,q);
  val(edtB.text,b,e);
  c:=a+b;
  edtC.text:=floattostr(c);
end;
```

Ha ellenőrizzük a feladatot, látni fogjuk, hogy nem tudunk hibaüzenetet generálni a pontosvesszővel, ponttal, sőt az értelmezhetetlen szöveggel sem, de még a gép számára értelmezhetetlen adattípushoz tartozó tartományt is túlléphetjük egy irreálisan magas érték beírásával. Látható ennek előnye, de ne feledkezzünk meg a hátrányáról sem! A véletlen hiba nem megfelelő eredményt fog jelezni a képernyőn.
A val függvény harmadik paramétere (példánkban q, illetve e) vizsgálatával saját hibakezelést is kezdeményezhetünk, ennek a változónak az értéke ugyanis megfelelő átalakítás esetén 0, minden egyéb esetben ettől eltérő értéket fog felvenni.

Önálló tevékenység sorozat:

Alakítsuk át egy kicsit még a felületünket, és próbáljuk meg önállóan elvégezni a képen látható gombokhoz tartozó műveleteket:

- A négyzetre emelésnél az első beviteli mezőnk tartalmát emelje a gép négyzetre. (Használhatjuk a beépített SQR függvényt)
- A gyök alá, szintén az első beviteli mezőben szereplő értéket tegyük (SQRT)
- A törléssel a beviteli mezők tartalmát töröljük, és adjuk a fókuszt az elsőre.

- A kilépés gomb a szokásos tevékenységet hajtsa végre, de a bezárás a felhasználó megerősítésével történhet csak meg. (emlékezzünk az OnCloseQuery-re!)

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Melyik nem része az Additional komponenspalettának?
   a., MaskEdit
   b., ScrollBox
   c., ComboBox
2. Mire alkalmas az Image komponens?
   a., Film állományok megjelenítésére
   b., A Form háttérképének tárolására
   c., Képek megjelenítésére
3. Milyen átalakításra képes az StrToFloat?
   a., Szövegből számmá.
   b., Szövegből egész számmá.
   c., Szövegből lebegőpontos számmá.
4. Melyik paranccsal állítható meg a programunk futása?
   a., Program Stop
   b., Program Wait
   c., Program Reset
5. Melyik menüből érhetjük el a fenti lehetőséget.
   a., Run
   b., Program
   c., Project

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. A runtime hibák ellen nem tudunk védekezni
   igaz
   hamis
2. A hibakezelésre a Delphi 4 lehetőséget kínál fel.
   igaz
   hamis
3. A programozói hibák 2 fő csoportba sorolhatók.
   igaz
   hamis
4. A Standard komponenspaletta része a DrawGrid.
   igaz
   hamis
5. A hibakezelések közé tartozik a Raise…Finally utasításszerkezet.
   igaz
   hamis

6. Az StrToFloat függvény képes az egész számokká történő átalakításra is.
   igaz
   hamis
7. A beviteli mező Caption tulajdonságának adhatunk közvetlenül is értéket.
   igaz
   hamis

III.

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Sorolja fel a Delphi kivételkezelésének lehetőségeit!
2. Részletezze a Try…Finally szerkezet felépítését!
3. Adja meg a szövegből számmá történő átalakítások lehetőségeit!
4. Mi a kivételkezelés szerepe?
5. Melyek a program futása során előforduló hiba csoportok?

A Delphi komponensei III.

A jelen fejezetben felsorolásszerűen megadjuk a Delphi további általánosan használt komponenspalettáinak elemeit, majd néhány alkalmazáson belül kiemelünk néhányat közülük a gyakorlati alkalmazásban történő bemutatás érdekében.

Win32 komponenspaletta:

A windows operációs rendszereiben megszokott kezelőszervek, komponensek csoportja.

- TabControl: Hasonlít a kartotékhoz, de csupán egyetlen oldala van
- PageControl: Kartoték csoport (jobbegér gombbal adható hozzá lap)
- ImageList: Képek csoportja, egy lista, vagy kép sorozat.
- RichEdit: Szövegszerkesztő.
- TrackBar: Csúszka, beállítások egyszerűbbé tételére alkalmazzuk.
- ProgressBar: Folyamatjelző.
- UpDown: Léptetőgomb.
- HotKey: Gyorsbillentyű definiálásához.
- Animate: AVI animációk megjelenítéséhez.
- DateTimePicker: Dátum/Idő lenyílólista.
- MounthCalendar: Dátum/Idő beállító naptár (havi bontásban).
- TreeView: Az elemek fastruktúrában történő megjelenítésére szolgál.
- ListView: Az aktuális útvonal elemeinek megjelenítésére szolgál.
- HeaderControl: Fejléc definiálása (amennyiben nem jó nekünk a szokásos)
- StatusBar: Állapotsor.
- ToolBar: Eszköztár.
- CoolBar: Állítható eszköztár létrehozásához.
- PageScroller: Az ablakon belül egy megjelenítő felületet biztosít (korlátoz).

System komponenspaletta:

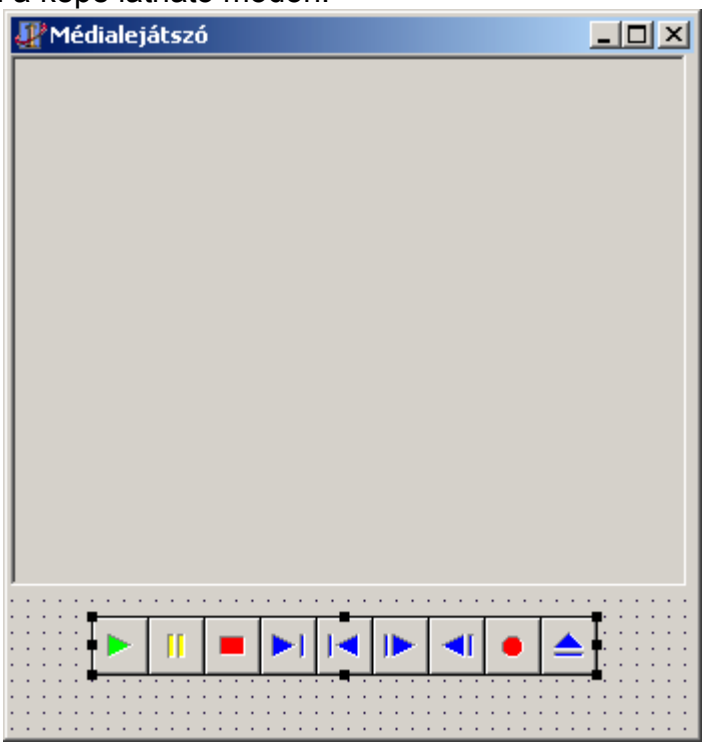
Itt találhatók a rendszerkomponensek:

- Timer: időzítő, a formok időhöz kötött eseményeihez.
- PaintBox: egy speciális terület elkülönítése rajzok készítéséhez.
- MediaPlayer: Média lejátszó (filmek, zenék megjelenítéséhez)
- OleContainer: OLE objektumok megjelenítője.
- DbeClientConv: A dinamikus adatcsere kliens oldali konverziójához
- DbeClientItem: A fenti adatcsere kliense.
- DbeServerConv: Az adatcsere szerveroldali konverziójához.
- DbeServerItem: A fenti adatcsere szervere.

Feladat: Médialejátszó készítése (Video)

A system komponensek segítségével készítsünk most el egy egyszerű média lejátszó alkalmazást. Hozzun létre a Delphi könyvtárban a Video alkönyvtárat.

1. Indítsuk el a Delphit, majd a megjelenő Form tulajdonságait állítsuk be: Name: frmVideo, Caption: Médialejátszó.
2. Mentsük az állományokat: Uvideo, Pvideo néven a Video könyvtárba.
3. Válasszuk ki System palettát, és kapcsoljuk be a MediaPlayert. Helyezzük el a formon, látni fogjuk, hogy egy egyszerű vezérlő gombsor jelenik meg. Helyezzük el ezt az form alsó részén.
4. Válasszuk most ki az OLEContainer-t is, és ezt helyezzük a gombsor fölé a formra. Ez lesz a „képernyőnk”, vagy megjelenítőnk, melyben az általunk hívott állomány futni fog. Rendezzük el az objektumokat nagyjából a képe látható módon.



Kérjünk mentést, és futtassunk. Látni fogjuk, hogy az alkalmazásunk inaktív, a gombok nem funkcionálnak. Ez természetes, hiszen nem rendeltünk állományt az alkalmazásunkhoz.

5. Tervezőnézetben keressük meg a MediaPlayer1-et (nem neveztük át), és a tulajdonságai között a FileName-et. Ez jelenleg üres, ide írjuk be a megnyitni kívánt állományt (teljes elérési útjával): c:\winnt\clock.avi.
   Ettől sajnos még nem lesz aktív a vezérlő (mentés után próbáljuk ki), át kell ugyanis még állítanunk az AutoOpen tulajdonságot is True-ra. Ekkor már működni fog, de külön ablakban nyitja meg az állományt.
6. Ha szeretnénk az általunk elhelyezett OleContener-t használni, akkor még állítsuk be a MediaPlayer1 Display tulajdonságát: OleContainer1.

Ezzel el is készült egy egyszerű médialejátszó, mely természetesen hagy némi kívánnivalót maga után: nem tudunk megnyitni közvetlenül állományt, az Ole konténerünk statikus mérettel dolgozik, és így tovább. Vegyük azonban észre, hogy néhány pillanat alatt sikerült egy lejátszó szoftvert készítenünk média állományainkhoz. Nézzünk egy további elemet a komponenspalettáról:

A rendszerkomponensek közül kiemelkedik még a Timer komponens, ennek megismeréséhez elkészítünk egy másik egyszerű alkalmazást.

Feladat: Futó reklámszöveg készítése (Reklam)

Hozzunk létre a delphi könyvtárunkban egy Reklam nevű almappát, majd indítsuk el a Delphit.

1. A form tulajdonságainál módosítsunk, a Name: frmReklam, Caption: Futó szövegek.
2. Kérjünk mentést a szokásos módon: Ureklam, Preklam.
3. Válasszuk ki a standard komponensek közül a Label komponenst, és helyezzük el a formon. Tulajdonságainál változtassuk meg a Name értéket: lblReklam-ra, a Caption pedig legyen: Ez egy reklám. Kérjük még le a Font tulajdonságot, és legyen a címkénk: Ms Sans Seriff, félkövér, 14-es, vörös.
4. Lépjünk a system palettára, és válasszuk ki a Timer komponenst, majd helyezzük el bárhol a formon (futásidőben nem fog látszani). Ezzel el is készültünk az alapokkal, nézzük a Timer működését.

A Timer komponens:

Ha a munkánk során szeretnénk olyan műveleteket végrehajtani, melyek időzítésekhez kötődnek, akkor tudjuk segítségül hívni a Timer komponenst. Míg korábban az Access esetében az űrlap rendelkezett időzítőre eseményvezérelt eljárással, és időzítő intervallummal, a delphi ezt a Timer komponenssel valósítja meg. Mi most a címkénket szeretnénk a képernyőn másodpercenként léptetni, hogy a statikus feliratból, futó reklámfelirat legyen. Ehhez az időzítőre azért van szükségünk, mert a léptetést időközönként kell megtennünk. Nézzük mi lesz a dolgunk:

5. Tervező nézetben kattintsunk duplán a formunkon lévő Timer komponensünkre, így rögtön az alapértelmezett (Timer) eseményhez jutunk a kódszerkesztőben. Ahhoz, hogy a címkénk lépjen a képernyőn, tudnunk kell, hogy a címkének van egy Left (bal oldal értéke) tulajdonsága. Ha ezt léptetjük másodpercenként mondjuk 10 pixelnyire, akkor elindul a feliratunk. Ehhez a kód viszonylag egyszerű:

lblreklam.left:=lblreklam.left+10;

Ha beírtuk a kódot, mentettünk, és futtatunk, látni fogjuk, hogy az élet nem ilyen egyszerű. Feliratunk szépen kisétál a képernyőről, az ablak alján megjelenő (és folyamatosan növekvő) görgetősávot hagyva maga után, s mi csak akkor láthatjuk újra, ha elég gyorsan utána tudunk tekerni a görgetősávval… A problémát természetesen mi magunk okoztuk, hisz nem rendelkeztünk a kódunkban arról, hogy mi történjen, ha elértük az ablak keretét. Fogalmazhatunk úgy is, hogy sikerült egy végtelen folyamatot, végtelen ciklust generálnunk… Nyilván az űrlap szélének elérésekor el kellene újra indítanunk a feliratot a kezdeti pozícióból. Javítsuk tehát a kódunkat:

6. Tervező nézetben hívjuk elő újra az előbb írt kódot, és módosítsuk a következőképpen:

```
procedure TfrmReklam.Timer1Timer(Sender: TObject);
begin
if lblreklam.left+lblreklam.width<frmreklam.width-10 then
       //ha még nem értük el az űrlap szélénél 10 pixellel előbb lévő pozíciót,
lblreklam.left:=lblreklam.left+10       //akkor léptetünk
else                                    //különben
lblreklam.left:=1;                      //a felirat kerüljön vissza
end;

end.
```

Ha mentettünk és futtattuk az alkalmazást, tapasztalni fogjuk, hogy már visszatér a kezdő pozícióba a címke, de a visszatérés előtt egy-egy pillanatra megjelenik a vízszintes görgetősáv a formon. Ha ezt nem szeretnénk látni, elegendő, ha a form tulajdonságlapján az AutoScroll tulajdonságot átállítjuk False-ra.
Ha tovább szeretnénk csinosítani a feliratot, akkor a kódunkat még kiegészíthetjük a színek megjelenítéséért felelős Color tulajdonságot, egy véletlenül generált RGB színnel, így a felirat még bennünket is meg fog tudni lepni:

lblreklam.font.color:=rgb(random(256),random(256),random(256));

Az utasítást természetesen az időzítő eseményhez írjuk, az elágazás után, a záró end (**end;**) elé, így az elágazásunktól függetlenül másodpercenként végrehajtásra kerül.

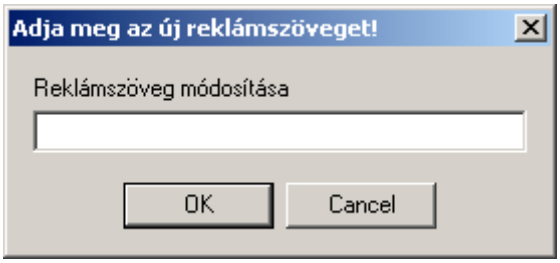
Az InputBox függvény:

Adjunk még lehetőséget a felhasználónak a statikus reklámfelirat megváltoztatására is:

7. Helyezzünk el a formon egy Button komponenst, a Name:btnSzoveg, a caption: Reklámfelirat módosítása.
8. Kattintsunk duplán a gombon, majd a megjelenő ablakba (btnSzoveg.Click eseménye) írjuk be a következő kódot:

```
procedure TfrmReklam.btnSzovegClick(Sender: TObject);
begin
lblReklam.Caption:=InputBox('Adja meg az új reklámszöveget!','Reklámszöveg módosítása','');
end;

end.
```

Mentsük a kódot, majd futtassunk, és nézzük meg az eredményt.



Mint látható, az InputBox függvény 3 paraméterrel rendelkezik: az első a paraméter bekérő ablak felhasználónak megjelenő szövege, a második a bekérő ablak fejrész felirata, a harmadik pedig a paraméterbekérő ablakban automatikusan megjeleníteni kívánt alapértelmezett szöveg. A fenti utasításunkkal megjelenítettünk a felhasználónak futásidőben egy paraméter ablakot, majd az oda írt adatot (string!) futásidőben átadtuk a címkénknek.

Ezzel is vannak azonban problémák, ha üresen hagyja a felhasználó a mezőt, vagy a cancel-t választja, nagyon elnéptelenedik az alkalmazásunk. Öröm az ürömben, hogy legalább hibaüzenetet nem küld a program, így ezzel most nem kell foglalkoznunk. Kikapcsolódásképpen nézzük tehát tovább a következő komponenscsoportokat.

A DataAccess, DataControls, Interbase és a Qreport komponenspaletta elemei a delphi adatbázis kezeléséhez szükségesek, és egy későbbi fejezetben tekintjük át őket részletesen. Az Internet, és a FastNet elemekkel jegyzetünkben nem fogunk foglalkozni, amennyiben szükségünk lesz rá, tanulmányozzuk a súgót.

A Dialogs komponenspaletta:

Ebben a palettában találhatóak meg a windows szabványos párbeszédablakainak megjelenítésére szolgáló komponensek.

- OpenDialog: a megnyitás párbeszédpanel.
- SaveDialog: a mentés párbeszédablaka.
- OpenPictureDialog: a képek megnyitásához.
- SavePictureDialog: a képek mentéséhez.
- FontDialog: A betűkészletek kezeléséhez.
- ColorDialog: a színkezelések eléréséhez.
- PrintDialog: a nyomtatáshoz.
- PrinterSetupDialog: a nyomtatók beállításához.
- FindDialog: a keresés ablakhoz.
- ReplaceDialog: a keresés-csere párbeszédablaka.

Az itt található valamennyi elem egy szabványos, a windowsban használt párbeszédablak megjelenítését teszi lehetővé. Végrehajtásuk során, a nevükkel hivatkozhatunk rájuk, és egy előugró ablakban megjelenik a kívánt elem, párbeszédablak.

Feladat: Dialógusablakok indítása (Dialog)

7. Hozzunk létre egy Dialog nevű alkönyvtárat a Delphi könyvtárunkon belül, majd indítsuk el a Delphit.
8. Az új alkalmazásunk Form tulajdonságát állítsuk be a következő módon:
    a. Name: frmDialog
    b. Caption: Párbeszédablakok indítása
9. Mentsük az alkalmazásunkat, Udialog, Pdialog néven.
10. Helyezzünk el egy OpenDialog panelt a formon. Ne változtassunk rajta semmit (így a neve OpenDialog1 lesz majd)
11. Helyezzünk el egy gombot is a formon, amivel elindítjuk a megnyitás panelt. A gomb tulajdonságainál állítsuk be a következőt: Name:btnOpen, Caption: Megnyitás.
12. Kattintsunk duplán, a gombunkon (vagy válasszuk az Events fül OnClick eseményét), és írjuk be a következő kódot a megjelenő Begin…End közé:

```
procedure TfrmDialog.btnOpenClick(Sender: TObject);
begin
opendialog1.Execute;
end;

end.
```

Ha mentettünk, és futtatunk, látni fogjuk az eredményt, és megértjük talán a Dialogs paletta jelentőségét is.

A Win 3.1 komponenspaletta

Ez a komponenspaletta a 3.1-es windows alá készítendő alkalmazások szabványos komponenseit tartalmazza. Természetesen valamennyi alkalmazásunkban használhatóak, nem csupán a régebbi windows felületen.

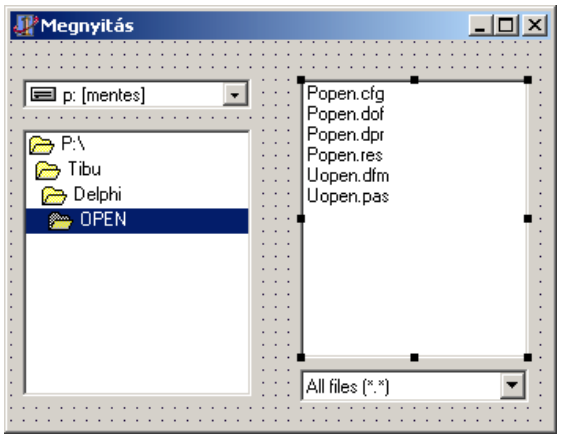
- TabSet: lapozó fülek létrehozására alkalmas
- Outline: elemek fastruktúrában történő megjelenítését teszi lehetővé.
- TabbedNotebook: Karton vezérlőelemet hozhatunk vele létre.
- Notebook: Egy több oldalas szöveges információk megjelenítésére alkalmas elem
- Header: Fejléc létrehozásához.
- FileListBox: állományok listázásához.
- DirectoryListBox: Könyvtárszerkezet listázásához.
- DriveComboBox: Meghajtó kiválasztásához.
- FilterComboBox: Állomány listák szűréséhez.
- DBLookupListBox: Adatbázisok adatainak listázásához.
- DBLookupComboBox: Adatbázisok lenyíló listáihoz.

Az utóbbi két komponenshez természetesen szükségünk van kiegészítő komponensekre is, melyeket az adatbázis-kezelésnél részletesen tárgyalni fogunk.

Feladat: Önálló megnyitás párbeszédpanel készítése (Open)

Hozzunk létre a delphi könyvtárunkban egy Open könyvtárat.

1. Indítsuk el a Delphit, majd a megjelenő form tulajdonságainál állítsuk be a következőket: Name: frmOpen, Caption: Megnyitás.
2. Mentsük az alkalmazásunkat: Uopen, Popen néven az Open könyvtárba.
3. A win 3.1 komponensek közül válasszuk ki a DriveComboBox elemet, és helyezzük a formunk bal felső sarkába. (nem állítunk rajta semmit)
4. Válasszuk ki a DirectoryListBox-ot is, és ezt helyezzük a DriveComboBox alá. (ennek az elemnek a tulajdonságaival sem foglalkozunk egyelőre)
5. Helyezzük a DriveComboBox mellé a FileListBox-ot, alá pedig a FilterComboBox-ot a képen látható módon:

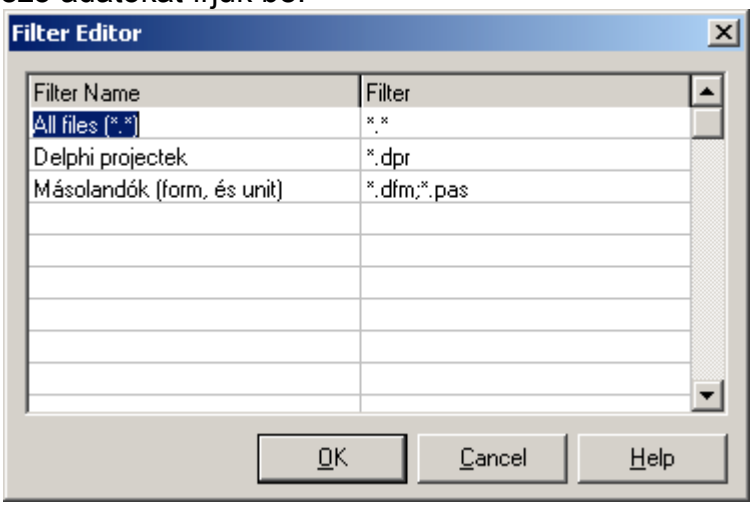


Ha mentettünk, és futtattuk az alkalmazást, láthatjuk, hogy a listázás a képernyőn statikus, az alapértelmezett könyvtárunk tartalmát fogjuk látni. Egyedül mindkét baloldalon lévő elem működik, de nincsenek hatással az alattuk, illetve mellettük elhelyezett mappastruktúrát, valamint állományokat kirajzoló elemekre. Korrigáljuk a hiányosságokat.

6. Először tervező nézetben keressük meg a DriveComboBox-ot, és válasszuk ki a DirList tulajdonságát. Ott egyetlen elem lesz (DirectoryListBox), rendeljük ezt hozzá, majd mentsünk és futtassunk.
7. Most tervező nézetben keressük meg a DirectoryListBox-ot, és válasszuk ki annak FileList tulajdonságát. Itt is csupán egyetlen elem látható (FileListBox1), rendeljük ezt hozzá, megint mentsünk, és futtassunk.

Tulajdonképpen már működik minden, de ebben a formában megkérdőjelezhető a FilterComboBox elhelyezése a lapon. A FilterComboBox-ot az állományok szűrésére használhatjuk, szűkítve ezzel az adott alkönyvtárban megjelenő elemeket, és megkönnyítve így a felhasználónak egy bizonyos csoport kiválasztását. Adjuk meg ezt a lehetőséget:

8. Tervező nézetben válasszuk ki a formról a filterComboBox elemet, majd keressük meg annak Filter tulajdonságát. Kattintsunk a tulajdonságra, majd az ott megjelenő … ikonra. Egy párbeszédablakot fogunk látni, ide a következő adatokat írjuk be:



Látható, hogy a szűrésnél egyszerre több állománytípus is megadható, ha tagoljuk őket a pontosvesszővel. Természetesen a lista bővíthető, számunkra a példához ez elég is lesz.

Bár úgy tűnik készen vagyunk, akik futtatják a munkát látni fogják, hogy a szűkítés sikertelen, hiába választanak ki bármilyen típust, minden állomány a képernyőn marad. Ennek oka az, hogy a szűrését felelős elem, még nem lett hozzákapcsolva az állománylistához.

9. Tervező nézetben keressük meg az előbb beállított FilterComboBox tulajdonságai között a FileList tulajdonságot, és állítsuk be az ott látható FileListBox1-et.

Elkészültünk, az alkalmazásunk működik, az objektumainkat hierarchikusan összekapcsolva egy teljes értékű, magunk készítette megnyitás ablakkal rendelkezünk.

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Melyik nem a Win32 komponenspaletta része?
   a., RichEdit
   b., MediaPlayer
   c., StatusBar
2. Mire alkalmas a DateTimePicker?
   a., Dátum/idő beállító naptár
   b., Rendszer idő komponens
   c., Dátum/Idő lenyíló lista
3. Hol található a Timer komponens?
   a., a Win32 panelen
   b., a System panelen
   c., a Dialogs panelen
4. A Form objektum melyik tulajdonsága felel a vízszintes görgetősáv automatikus megjelenítéséért?
   a., HorzScrollBar
   b., VertScrollBar
   c., AutoScroll
5. Milyen paraméterekkel rendelkezik az InputBox függvény?
   a., csak szövegekkel
   b., szöveggel, és integerrel
   c., szöveggel, és súgó környezetazonosítóval.
6. Hol található a NoteBook komponens?
   a., a Dialogs palettán.
   b., a win32 palettán.
   c., a win 3.1 palettán.
7. A FilterComboBox-ban egyszerre több állománytípus is megadható
   a., vesszővel elválasztva.
   b., pontosvesszővel tagolva.
   c., kötőjellel elválasztva.
8. A címkéinket futásidőben is igazítani tudjuk vízszintesen a kereten belül
   a., a Right tulajdonság állításával
   b., az Align tulajdonság állításával
   c., a Left tulajdonság állításával

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. Csak a beépített komponenseket használhatjuk.
   igaz
   hamis
2. Az objektumok igazítását a formon pixelben adhatjuk meg.
   igaz
   hamis
3. A ToolBar a System palettán található.
   igaz
   hamis
4. Az RGB függvény segítségével képesek vagyunk önálló színeket definiálni a háttérben futó kódban is.
   igaz
   hamis
5. Az InputBox függvény információk megjelenítésére használható.
   igaz
   hamis
6. A PrinterSetupDialog ablak a nyomtatás párbeszédpanelt jeleníti meg.
   igaz
   hamis
7. A FileListBox az állományok megjelenítésére alkalmas.
   igaz
   hamis
8. Ha nem állítunk be semmit a MediaPlayer komponensen, akkor egy önálló ablakban nyitja meg a lejátszandó állományt.
   igaz
   hamis
9. A Win 3.1 paletta tartalmaz adatbázis kezeléshez használható elemeket is.
   igaz
   hamis
10. A PaintBox a WIN32 palettán található.
    igaz
    hamis

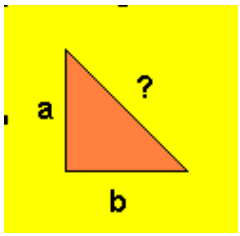
Összetett alkalmazások fejlesztése.

Ebben a fejezetben az eddigi ismereteket felhasználva igyekszünk egy viszonylag összetett alkalmazást készíteni, miközben néhány friss ismeretre is szert tehetünk, végül két egyszerű alkalmazás segítségével a listák kezelésével is megismerkedünk

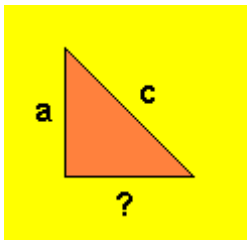
Feladat: Pitagoras tételének kidolgozása (Pita)

A feladatban egy vizuális felületen fogjuk Pitagoras tételét a felhasználótól bekért adatoknak megfelelően kiszámítani. Ehhez készítsük el a Delphi könyvtárunkban a Pita nevű alkönyvtárat.

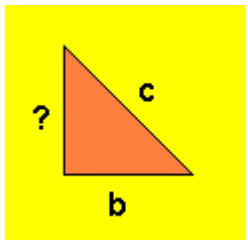
1. Indítsuk el a Paint programot, majd rajzoljunk meg három képet, melyet fel fogunk használni a programunk látványosabbá tételéhez. Mentsük őket a megadott néven a Pita könyvtárunkba:



c.bmp



b.bmp



a.bmp

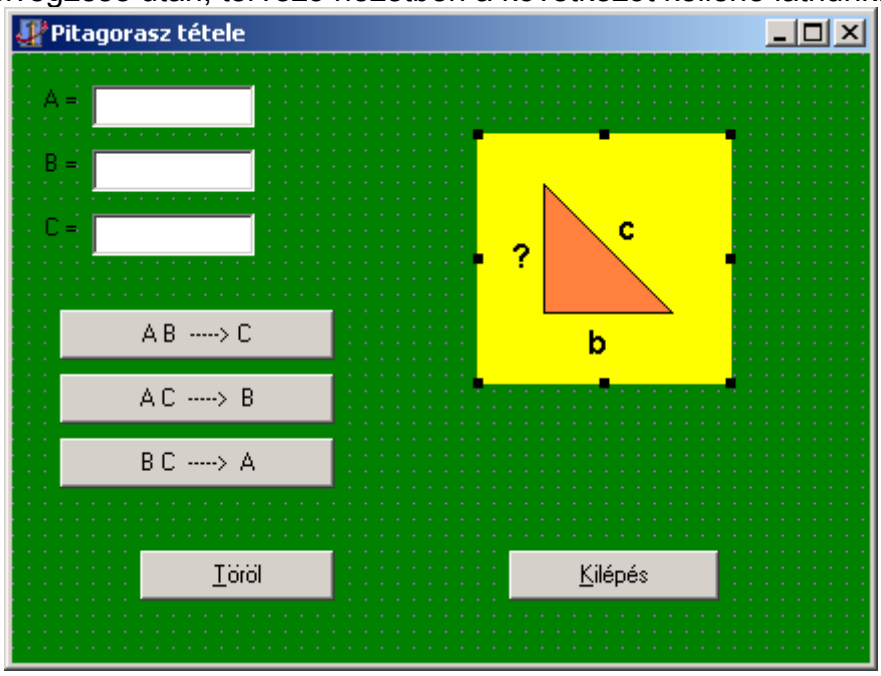
2. A Delphi indítása után a Form Name: frmPita, a Caption: Pitagoras tétele, a Color: clGreen. Amennyiben más színt szeretnénk, akkor duplán kattintva a színek lenyíló listáján egy korábbról már ismerős felületből tudunk választani.
3. Kérjünk mentést: Upita, Ppita a Pita könyvtárba.
4. Helyezzünk el 3 Edit komponenst a form bal felső részére, melyek neve fentről lefelé haladva: edtA, edtB, edtC. A Text tulajdonságukat töröljük ki.
5. Helyezzünk el az Edit-ek elé 1-1 Label komponenst, melyeknek a Caption mezői fentről lefelé: A=, B=, C=. A nevük: lblA, lblB, és lblC.
6. A címkék, és a mezők alá helyezzünk el 3 gombot egymás alá. Nevük (megint fentről lefelé): btnC, btnB, és btnA, feliratuk: „AB→C”, „AC→B”, „BC→A”.
7. A három gomb alá a form alsó részére rakjunk még egymás mellé 2 gombot. A bal oldali neve: btnTorol, Caption: &Törlés. Mellette lévő neve: btnKilep, Caption: &Kilépés.

Kép objektumok elhelyezése a formon

Elkészítettünk korábban 3 ábrát, melyeket a formunkon szeretnénk megjeleníteni, ezzel is gazdagítva a képi megjelenítést. Ehhez az additional paletta Image objektumára van szükségünk:

8. Keressük meg az Image objektumot, majd helyezzük el a formunkon a beviteli mezők melletti területen. Keressük meg a Name tulajdonságát, és írjuk át: imgBC-re. A kép betöltéséhez kattintsunk duplán a formon lévő komponensre, majd a megjelenő párbeszédablakban a „Load” gomb segítségével tallózzuk ki az „a.bmp” képet. Hagyjuk jóvá az „OK” gombbal a betöltést. Igazítsuk az Image méretét a képünk méretéhez.

A művelet elvégzése után, tervező nézetben a következőt kellene látnunk:



Sajnos jelenleg csak egyetlen képünk van, így szükségünk lesz még 2 Image-re, melyek az eredeti Image méretével egyeznek, mivel a képeket egymásra szeretnénk helyezni, majd a láthatóságukat úgy módosítani, hogy éppen melyikre van szükségünk. Ezt legegyszerűbben úgy tehetjük meg, ha kijelöljük az imgBC komponensünket, és az Edit menüből kérjük a másolást (Copy), majd a beillesztést (Paste). Ekkor az eredeti Image méreteivel egyező objektumot fogunk kapni. Nevezzük át: imgAB-re, majd töltsük be a hozzá tartozó képet (c.bmp) duplát kattintva az image-n. Már csak rá kell igazítanunk az előző image-re az új image-t, és jöhet az utolsó Image. Mivel vágólapon van az előző másolás eredményeként az első Image, így most csak kérjünk az Edit menüből beillesztést (Paste). Nevezzük át az új image-t: imgAC-re, és töltsük be az utolsó (b.bmp) képet, majd igazítsuk rá az alatta lévő image-re. Így elhelyeztünk egymásra 3 Image objektumot. Futtatáskor természetesen csak egyet, az utolsó elkészítettet fogjuk látni, mivel ez eltakarja a többit. De tulajdonképpen indításkor nem is szeretnénk egyiket sem látni, így akár át is állíthatnánk a Visible tulajdonságukat False-ra, de ezt is inkább a kódban tesszük majd.
Az alapok elkészültek, nézzük mi a teendőnk. A legegyszerűbb a kilépés, írjuk meg a gombhoz tartozó close; utasítást, majd mentsünk, és futtassunk. (Csak emlékeztetőül: ne felejtsük, hogy a kilépést célszerű megerősítéssel összekötni, amit a formunk OnCloseQuery eseményéhez kellene írnunk)
A következő feladat a törlés gomb. Funkciója: az esetlegesen elvégzett számításokhoz tartozó valamennyi mező tartalmát alapra állítani (törölni), és a képek láthatóságát megszüntetni. Mivel nem tudjuk eldönteni közvetlenül, hogy éppen melyik látszik, (ami attól függ, hogy melyik művelete(ke)t hajtotta végre a felhasználó) így egyszerre valamennyiét eltüntetjük majd.

9. Menjünk a törlés gombhoz, és kattintsunk rajta duplán. A kódszerkesztőben írjuk be a beviteli mezők törlésére szolgáló kódot.

Komponensek láthatóságának változtatása futásidőben

A beviteli mezők törlése után, az Image objektumaink láthatóságát kell megváltoztatnunk, pontosabban láthatatlanná kell őket tennünk. Ha megnéztük az objektum-kezelőben az Image tulajdonságait, akkor láthattuk, hogy rendelkezik egy Visible tulajdonsággal, mely igaz-hamis (true-false) értékeket vehet fel. Tehát a kódunk: imgAB.Visible:=False;
Ezt kell még a másik két objektumhoz is hozzárendelnünk, és elkészültünk a törlés gombbal:

```
procedure TfrmPita.btnTorolClick(Sender: TObject);
begin
     edtA.text:='';
     edtB.text:='';
     edtC.text:='';
     imgAC.visible:=false;
     imgBC.visible:=false;
     imgAB.visible:=false;
end;
```

Kódok újrahívása

Az elkészült kódunkat használhatjuk még az űrlapunk megnyitásakor is, így nem kell ott külön még egyszer láthatatlanná tennünk a képeinket. Hívjuk meg tehát a form OnCreate eseményét, majd a Begin…End közé írjuk be az előző kódunk futtatásának végrehajtásáért felelős kódot:

```
procedure TfrmPita.FormCreate(Sender: TObject);
begin
frmPita.btnTorol.Click();
end;
```

Az utasításunk a form létrehozásakor (indításakor) végrehajtja a töröl gombunk kattintásra megírt eljárását. Jöjjenek a számításért felelős gombok.
Az első gombunk (btnC) a (feltételezetten) meglévő A, és B értékekből számítaná ki Pitagoras tételének segítségével a C oldal értékét, miközben láthatóvá teszi a számításunkhoz tartozó képet. Ez utóbbi feladat tűnik a legegyszerűbbnek, csak láthatóvá kell tennünk az image-t (imgAB), melyben a képünk található.
Van azonban egy apró probléma. Ha ezt a műveletet nem először hajtja végre a felhasználó, hanem közben már egy másik műveletet is kezdeményezett, akkor annak a képe is látható. Mivel a felhelyezés sorrendje szerint takarják egymást a képek, így nem elegendő egyetlen képpel foglakoznunk, valamennyit be kell állítanunk, függetlenül attól, hogy éppen láthatatlanok voltak-e vagy sem. A kódban, tehát míg az imgAB-t láthatóvá tesszük, addig a másik két képet láthatatlanná is kell tennünk:

```
imgAB.visible:=true;
imgAC.visible:=false;
imgBC.visible:=false;
```

A számításainkhoz szükségünk lesz azokra a változókra is, melyekben az értékeket fogjuk tárolni. Ezeket a Begin elé deklaráljuk: a,b,c:double;

Beviteli mezők ürességének vizsgálata

A műveletünk elvégzése előtt meg kell vizsgálnunk, hogy nem hagyta-e üresen az szükséges alapadatokat a felhasználó, vagyis adott-e meg értékeket az A, és a B értékeket tároló beviteli mezőkbe. Az üresség vizsgálata itt megint nem elég, hisz adhatott akár valami értelmetlen adatot is a mezőkbe, tehát jönnek a hibakezelések, amihez két további (integer) változóra is szükségünk lesz:

```
procedure TfrmPita.btnCClick(Sender: TObject);
var a,b,c:double;
    q,e:integer;
begin
     imgAB.visible:=true;
     imgAC.visible:=false;
     imgBC.visible:=false;
     if (edtA.text='') then begin
        showmessage('Adja meg az A oldal értékét!');
        edtA.setFocus;
        exit;
     end;
     if (edtB.text='') then begin
        showmessage('Adja meg a B oldal értékét!');
        edtB.setFocus;
        exit;
     end;
     val(edtA.text,a,q);
     val(edtB.text,b,e);
end;

end.
```

Az utolsó két sorunk tartalmazza a tényleges beolvasást.
Ezt követően még mindig ellenőriznünk kell, hogy a már biztosan számot tartalmazó értékek megfelelnek-e egy háromszög oldalainak kiszámításához, vagyis újabb hibakezelések következnek. A kódot természetesen az átalakítások utáni sorban folytatjuk:

```
if (a=0) then begin
   messagedlg('A=0, nem háromszög!',mtinformation,[mbok],0);
   edtA.setFocus;
   exit;
end;
if (a<0) then begin
   messagedlg('A<0, negatív!',mtinformation,[mbok],0);
   edtA.setFocus;
   exit;
end;
if (b=0) then begin
   messagedlg('B=0, nem háromszög!',mtinformation,[mbok],0);
   edtB.setFocus;
   exit;
end;
if (b<0) then begin
   messagedlg('B<0, negatív!',mtinformation,[mbok],0);
   edtB.setFocus;
   exit;
end;
```

Végül végrehajthatjuk a számítási műveletet, majd egy visszaalakítás után átadjuk az eredményt a harmadik beviteli mezőnek:

```
c:=sqrt(sqr(a)+sqr(b));
edtc.text:=floattostr(c);
```

Mint látható, a tényleges tevékenységünk csupán néhány utasítás lett volna, amihez képest a végső kódunk meglehetősen terjedelmes lett. A feladatban azt igyekeztük kihangsúlyozni, hogy nem elegendő a feladat felületes végrehajtása, körültekintően foglakoznunk kell a hibakezelésekkel is.

A fenti gomb eseményét alapul véve készítsük most el önállóan a másik két gomb eseményét is, majd teszteljük alkalmazásunkat, hogy a várt módon működik-e. Amennyiben mindent rendben találtunk ne felejtkezzünk meg a mentésekről sem.

(A körültekintő tesztelés természetesen azt jelenti, hogy igyekszünk minden részletre kiterjedően tesztelni. Ha az előző feladatban nem találtunk hibát, akkor próbáljuk ki a programot úgy, hogy a B értéke mondjuk 4, a C értéke pedig 3, és próbáljuk így ki a BC→A gombunkat…Majd szomorodjunk el, és gondolkodjunk a megoldáson, és ne felejtsük: a programozás alapja a matematika…)

Feladat: Lista elemek kezelése (Helyez)

A következő egyszerű alkalmazásban a Delphi Lista komponensének kezelését fogjuk elvégezni. Ehhez hozzuk létre a delphi könyvtárunkban a Helyez almappát.

1. Indítsuk el a Delphit, a megjelenő form Name: frmHelyez, Caption: Lista elemeinek áthelyezése, Color: clGreen.
2. Kérjünk mentést: Uhelyez, Phelyez.
3. Keressük meg a Standard komponenspalettán a ListBox elemet, majd helyezzük el a formon a baloldalon középre. Name: lstForras. Válasszunk ki egy másik ListBox-ot, azt helyezzük a másik mellé a jobb szélre, középre. Name: lstCel.
4. Helyezzünk el öt gombot az űrlapunkon a két lista közé, fentről lefelé haladva. Az első neve: btnBalrol, Caption: „>” A következő neve: btnBalmind, Caption: „>>”. Alatta: btnJobbrol, Caption: „<”, majd alatta: btnJobbmind, Caption: „<<”. Az utolsó gombunk a kilépés, Name: btnKilep, caption: &Kilépés.
5. Töltsük még fel a bal oldali listánkat elemekkel. Kattintsunk az első (lstForras) elemünkre, majd keressük meg a tulajdonságai között az Items tulajdonságot, és ott kattintsunk a (TStrings)… három pontra. A megjelenő párbeszédpanelbe vigyünk fel néhány nevet, a nevek után egy-egy Enter-t ütve soronként: Andrea, Anita, Beáta, Cecília, Diána.
   Az OK gombbal zárjuk be a String List Editor ablakot.

Ha mindent megfelelően hajtottunk végre, a következő képet fogjuk látni:

Amennyiben mégsem ezt látnánk, módosítsunk. Természetesen a gombok mérete, elrendezése ízlés szerint alakítható. A következő lépések a listaelemek kezelésének utasításai lesznek. Nézzük a Delphi miként képes nyilvántartani a listákat. Első kódunkat a legfelső gombhoz (btnBalrol) írjuk meg, melynek feladata, egy kiválasztott elem balról történő áthelyezése a jobboldali listába.

6. Kattintsunk duplát a legfelső gombon, majd a kódszerkesztőben a Begin…End közé írjuk be a következő sorokat (természetesen a komment kihagyható):

```
procedure TfrmHelyez.btnBalrolClick(Sender: TObject);
begin
     lstCel.Items.Add(lstForras.Items.Strings[lstForras.ItemIndex]);
     //a bal oldali lista elemét hozzáadjuk a jobb oldali listához
     lstForras.Items.Delete(lstForras.ItemIndex);
     //majd a bal oldali elemet töröljük (áthelyezés!)
end;
```

A következő kód a bal oldali valamennyi elemet áthelyezi a jobb oldalra, majd a bal oldali lista tartalmát ürítjük.

7. Kattintsunk duplát a btnBalmind gombon, és a kódszerkesztőbe írjuk be a következő kódot:

```
procedure TfrmHelyez.btnBalmindClick(Sender: TObject);
begin
     lstCel.Items.Addstrings(lstForras.Items);
     lstForras.Items.Clear;
end;
```

Látható a kódban a lista valamennyi elemére történő hivatkozás (Addstrings), és a lista tartalmának ürítése, a törlés (Clear).

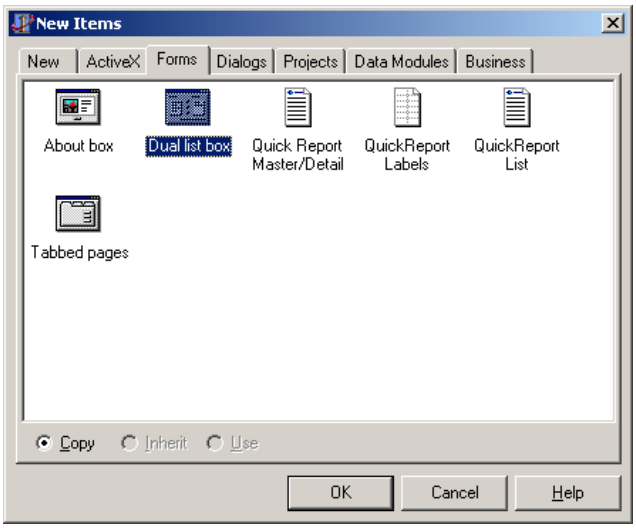
A fenti kódoknak megfelelően önállóan oldjuk meg, hogy a másik két gombunk is működjön, mentsünk, majd teszteljünk.
Remélhetőleg minden rendben lévőnek tűnik, adjunk egy elemet azonban a jobb oldali listába, majd nyomjuk meg kétszer a btnJobbrol gombunkat… (Elképzelhető, hogy ezt sem kell megtennünk, elég ha az induláskor megpróbálunk áthelyezni balról

egy elemet a jobb oldali listába úgy, hogy nem jelölünk ki semmit a bal oldalon lévő listából)…

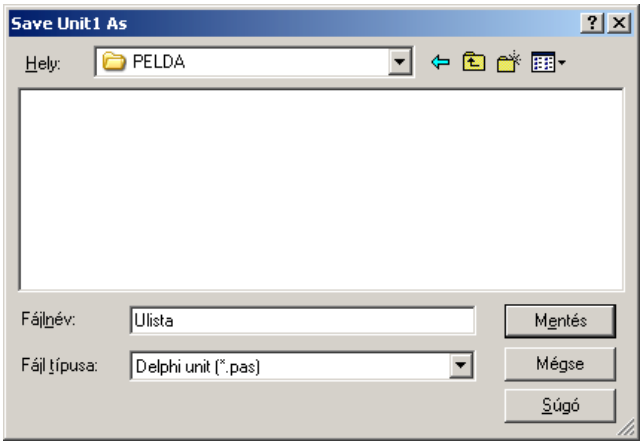
Ugye ismerős a képernyő?
A probléma a tömbön túli indexelésre, mint programozói hibára vezethető vissza. Esetünkben megpróbáltuk a jobb oldalon lévő lista elemét áthelyezni balra, de a jobb oldalon nem volt elem. Ha már indításkor hibaüzenetet kaptunk, akkor a baloldalon nem volt kiválasztott elem (ezt a hibát nem feltétlenül fogjuk tapasztalni azért, mert a fókusz az aktív bal oldali listán áll, vagyis az első elem kiválasztott lehet…) A probléma nyilvánvaló megoldása az, hogy minden egyesével történő áthelyezés (bal oldali, jobb oldali) esetén, meg kell vizsgálnunk, hogy van-e kiválasztott (Selected) elem. Ha nincs, akkor nem teszünk semmit, vagy hibaüzenetben közöljük, hogy válasszon ki a felhasználó egy elemet. Az elágazásunk másik ágába pedig maga az áthelyezés kerülhet.
Bár elkészíthetnénk a hibakezelést közösen is, most inkább ismerkedjünk meg a Delphi egy újabb lehetőségével. Hogy ne felejtsük el később majd javítani a hibát, helyezzünk el egy kommentet a btnBalrol, és a btnJobbrol gombokhoz, melyben jelezzük a tömbön túli indexelést kiválasztatlan elemek esetén. Ezt beírva mentsük az alkalmazásunkat, majd zárjuk be azt a File menü Close All utasításával.

A Delphi beépített példaalkalmazásai

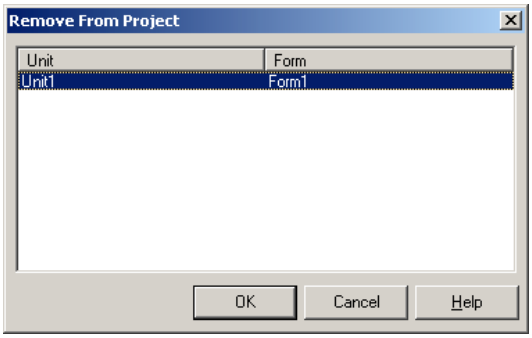
Lehetőségünk van néhány példaalkalmazás megtekintésére a Delphin belül, melyeket a fejlesztők mellékeltek a Delphi programhoz, ezzel is segítve az otthoni fejlesztők munkáját. A File menüben a New gombra kattintva, a megjelenő párbeszédablakból válasszuk most ki a példaalkalmazásunk mását: lépjünk a New Items ablakban a Forms fülre, és a belső ablakban válasszuk ki a DualListBox-ot dupla kattintással:



Mint látható, azonnal megjelenik a Form, és a háttérben ott a kód is, de van egy aprócska probléma: nem tudjuk futtatni az alkalmazást sem az eszköztárról, sem a menüből, de a szokásos módok egyikén sem. Tulajdonképpen ez bennünket nem is izgatna most annyira, hisz a kód hozzáférhető, tehát meg tudjuk tekinteni, ki tudjuk másolni is. De mi is lehet a baj?
Az, hogy nincsen project-ünk. Az előző alkalmazásunkat bezártuk a Close All utasítással, és most a Forms fülről választottunk egy elemet. Értelemszerűen így nincsen projectállományunk. Mit tehetünk most?
Mentsük el a Delphi Unit-ját a Save All utasítással. A felkínált párbeszédpanelen belül legyen az állomány neve Ulista.pas, és a felső lenyíló listából keressük meg Delphi almappánkat, majd hozzuk benne itt létre a PELDA nevű almappát. Lépjünk be a mappába, és ha mindent rendben találtunk (fent az új könyvtárunk neve, lent Ulista.pas), akkor kérjük a mentést:



Látható, hogy mentés után nem reklamált a Delphi a project mentéséért, így létre kell hoznunk a projectet. Menjünk a File menü New Application lehetőségéhez, és válasszuk ki azt. Az eredmény egy új alkalmazás, új formmal. Bárhol keresnénk, a másik formunk már nincs itt, amiben most dolgozunk egy teljesen új alkalmazás. Tüntessük el a megjelent üres formot, menjünk a Project menü Remove From Project lehetőségéhez, és az egyetlen Unit1 unitunkat távolítsuk el a párbeszédablak segítségével:



Ne lepődjünk meg az ürességen, az előbb az volt a baj, hogy nem volt project, most az a baj, hogy van project, de se unit nincsen, se pedig a hozzá tartozó kód. Már nincs más dolgunk, mint a háttérből előcsalnunk az előzőleg elmentett formot. Menjünk újra a Project menüpontra, most válasszuk, az Add To Project lehetőséget, majd a megjelenő párbeszédpanelben tallózzuk ki az előzőleg elmentett PELDA könyvtárunk Ulista.pas állományát. És minden rendben, futtatható az alkalmazás. Teszteljük, majd vizsgáljuk meg tüzetesen a kódot. A vizsgálódás végén kérhetjük még a project mentését (Save All), Plista néven.
Visszatérhetünk a saját listánkhoz, és megkísérelhetjük önállóan javítani annak hibáit.

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. A Form objektum melyik tulajdonsága alkalmas a háttérszín beállítására?
   a., BGColor
   b., BackGroundColor
   c., Color
2. Melyik komponens alkalmas képek megjelenítésére?
   a., Picture
   b., Image
   c., Paint
3. Melyik tulajdonság alkalmas a láthatóság beállítására?
   a., Show
   b., Visible
   c., Hide
4. Mit jelent a „List index out of bounds(0)” hibaüzenet?
   a., A lista üres
   b., A lista eleme nincsen kiválasztva
   c., Tömbön túli indexelést

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. Egy Image objektum egy időben, csak egy kép betöltésére, és megjelenítésére alkalmas.
   igaz
   hamis
2. A kódban megírt eljárásainkat bármikor, bárhonnan meg tudjuk hívni.
   igaz
   hamis
3. A Formok háttérszínénél csak a lenyíló listából tudunk színt választani.
   igaz
   hamis
4. Egy lista valamennyi elemének egy lépésben való törléséhez a Delete utasítást használhatjuk.
   igaz
   hamis
5. A Delphi projectet, a Run menüpont Add To Project menüjével bővíthetjük meglévő korábbi kódjainkkal.
   igaz
   hamis

III.

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Mire alkalmas a Remove From Project menüpont?
2. Milyen hibákat kell kezelnünk Pitagoras tételének kidolgozásakor?
3. Hogyan hívhatjuk meg a már korábban megírt kódunkat?
4. Mi a feladata az Image objektumnak?
5. Milyen tulajdonsággal adhatunk elemeket egy listához?

IV.

KÉREM OLDJA MEG A KÖVETKEZŐ FELADATOT!

1. Készítse el a saját Névjegy ablakát, felhasználva a Delphi New Formjának „About box” lehetőségét. Az alkalmazást mentse a delphi könyvtárban létrehozandó NEVJEGY mappába Unevjegy, Pnevjegy néven. Ügyeljen a névkonvenciókra, és írja felül a címkéket saját ízlése szerint

Többablakos (MDI) alkalmazás.

Az alkalmazások kialakításának lehetőségei a Delphiben.

A Delphivel történő fejlesztés kezdetén el kell döntenünk, hogy az elkészítendő munkánk milyen jellegű lesz. Alapesetben három lehetőség közül választhatunk:

Single Document Interface:

Ez az un egyablakos alkalmazás. Alapértelmezetten a Delphi ezt a lehetőséget kínálja fel egy alkalmazás fejlesztésének kezdetén. Az SDI kifejezés arra utal, hogy alkalmazásunk egyetlen ablakot fog használni, mindaz, amit feladatként megfogalmaztunk az alkalmazásunk számára, elfér egyetlen felületen. Az SDI alkalmazások legtipikusabb példája a Paint rajzoló program, melynek az a jellegzetessége, hogy nem képes egyszerre több kép (ablak) kezelésére.

Multiple Document Interface:

Ez a többablakos alkalmazás, az a felület, melyben a programunk rendelkezik egy főablakkal, és a benne végrehajtandó feladatok alablak(ok)ban, vagy gyermekablak(ok)ban jelennek meg. A többablakos alkalmazás tipikus példája a Word szövegszerkesztő, mely egyszerre több állomány kezelésére képes, természetesen akár különálló ablakokban kezelve azokat.

Egyéni alkalmazás felület:

Természetesen lehetőségünk van arra is (a Delphi elég rugalmas ezen a téren), hogy a két lehetőségen kívül, tovább bonyolítsuk a dolgot, és olyan összetett alkalmazást hozzunk létre, melyben mindkét felületet kombinálva (akár oda-vissza variálva) használjuk fel azokat. Általában elmondható azonban, hogy erre nem igazán van szükség, egy alkalmazás fejlesztése során már az elején eldönthető, hogy az alkalmazás melyik (MDI, SDI) csoportba fog tartozni.

Eddigi munkáink mindegyike kivétel nélkül un egyablakos, vagyis SDI alkalmazás volt. Ebben a fejezetben összefoglalva eddigi ismereteinket egy több ablakkal rendelkező menüvezérelt alkalmazás fejlesztését fogjuk végrehajtani, néhány újabb elem megismertetésével együtt.

A tanulmányunk kezdetén volt már szó a korábbi munkáink helyreállításának lehetőségéről, ott kiemeltünk két fontos állományt a Delphi állománytípusai közül:

- A Delphi kódját tartalmazó, formonként létrejövő PAS állományt,
- és a komponensek segítségével fejlesztett Form kódját tartalmazó DFM állományt.

Ha szeretnénk korábbi munkáinkat felhasználni, illetve azokat átdolgozni, akkor erre a két állományra mindenképpen szükségünk lesz (feltéve, ha nem akarjuk újraírni a kódunkat, és újra elkészíteni az űrlapunkat).

Feladat: Többablakos alkalmazás készítése eddigi munkáink felhasználásával (MDI)

Hozzunk létre egy MDI alkönyvtárat a Delphi könyvtárunkban, majd másoljuk be ebbe a könyvtárba valamennyi eddig elkészített munkánk PAS, és DFM állományát a Dialog kivételével (hisz az csupán egy egyszerű párbeszédablak). Erre azért van szükségünk, mivel az alkalmazásunk az eredeti egyablakos alkalmazáshoz képest most többablakosként fog működni. Ha az eredeti könyvtárban módosítanánk a PAS, és DFM állományt, akkor az eredeti alkalmazásunk működésképtelenné válna, egy hibaüzenettel reagálna eredeti helyén történő futtatási kísérletünkre. Természetesen ettől az EXE állomány működne továbbra is, de fejleszteni már nem tudnánk az eredeti projectet. Ezzel a másolással, most a másolt kódokhoz fogunk csak hozzányúlni, így az eredeti változatlan marad.
Ha átmásoltuk az eddigi állományainkat az MDI könyvtárba, ott a következőknek kellene szerepelni:

Ha ellenőriztük, hogy valamennyi állományunk megvan (beleértve az előző alkalommal feladatként önállóan végrehajtandó névjegy alkalmazás állományait is), akkor indítsuk el a Delphit.
Ahhoz, hogy a meglévő egyablakos (SDI) alkalmazásainkat egyetlen főablakban használhassuk, létre kell hoznunk a keretalkalmazást. Ehhez a most megnyílt Form1-et fogjuk felhasználni, projectként pedig az aktuális új projectet. Ez fogja összefogni a teljes alkalmazást.

1. A form tulajdonságainál a Name: frmFo, ezzel jelezzük annak főablak voltát. A Caption legyen: Összetett MDI alkalmazás. Még egy fontos teendőnk van: a form tulajdonságainál keressük meg a FormStyle-t, (eddig ezzel nem kellett foglalkoznunk) és állítsuk át fsMDIForm-ra. Ezzel tájékoztattuk a delphit arról, hogy az ablakunk egy többablakos alkalmazás keretablaka lesz.
2. Kérjünk mentést a Save All-al, a Unit: Umdi, a project Pmdi néven kerüljön az MDI könyvtárunkba.

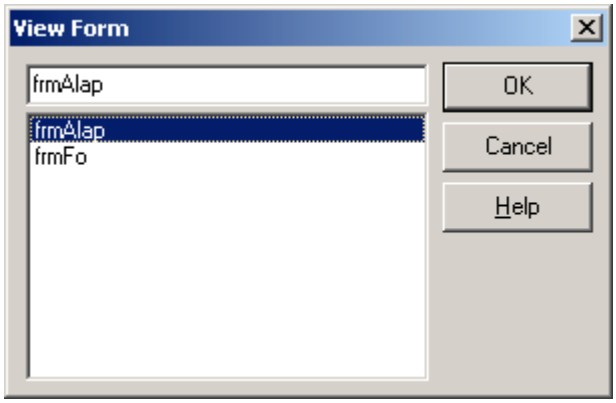
Ha futtatunk, látni fogjuk, hogy ablakunk képe egy kissé eltér az eddig megszokott ablakoktól, a keret belső része süllyesztett. Ettől függetlenül természetesen lehetőségünk van az alkalmazást egyablakosként fejleszteni, bár akkor nem lett volna értelme a formot MDIForm-ként beállítani.

Korábbi alkalmazás beépítése MDI formba

Nézzük, hogyan használhatjuk fel korábbi alkalmazásainkat. Erre már volt példa:

3. Válasszuk a Project menüpont, Add To Project almenüjét, és a megjelenő ablakból adjuk hozzá projectünkhöz a 4alap.pas állományt.

Szemmel láthatólag nem történt semmi, és ha mentés után futtatunk, akkor sem tapasztalunk semmi változást. Ennek oka, hogy a keretalkalmazásunkhoz nem kapcsolódik szervesen a projecthez rendelt alapműveleteket végző alkalmazásunk kódja. Akkor nem történt semmi? Dehogynem, ha megnézzük a View menü Forms almenüjét, akkor látni fogjuk, hogy a formunk a project része. Válasszuk ki az frmAlap formot (kattintsunk rá duplán).



Azonnal meg fog előttünk jelenni az általunk fejlesztett alkalmazás formja. Hogy integráljuk a főablakba, nem kell mást tennünk, mint megkeresve az frmAlap FormStyle tulajdonságát, átállítanunk azt fsMDIChild-ra.
Miért Child? Mert egy többablakos alkalmazás nyilván alapesetben csak egy főablakkal (MDIForm) rendelkezik, és a beépülő ablakaink annak gyermekablakai (MDIChild) lesznek majd. Győződjünk meg róla, hogy sikerült integrálni a formot, mentsünk, és futtassunk.

Memóriaterület felszabadítása, a gyermekablak bezárása

A formunk megjelent, és minden funkciója működik is, egyetlen apróságot kivéve, ez pedig a bezárás… A gyermekablakot a főablakon belül nem tudjuk bezárni, hisz most integráltuk a főablakunkba, így az csak a főablak bezárásakor fog eltűnni. Mit tehetünk tehát?

4. Menjünk tervező nézetben az frmAlap form Events lapjára, és keressük meg az OnClose eseményt. Kattintsunk duplán a mezőbe, majd a megjelenő ablakba írjuk be a következő kódot a Begin…End közé:

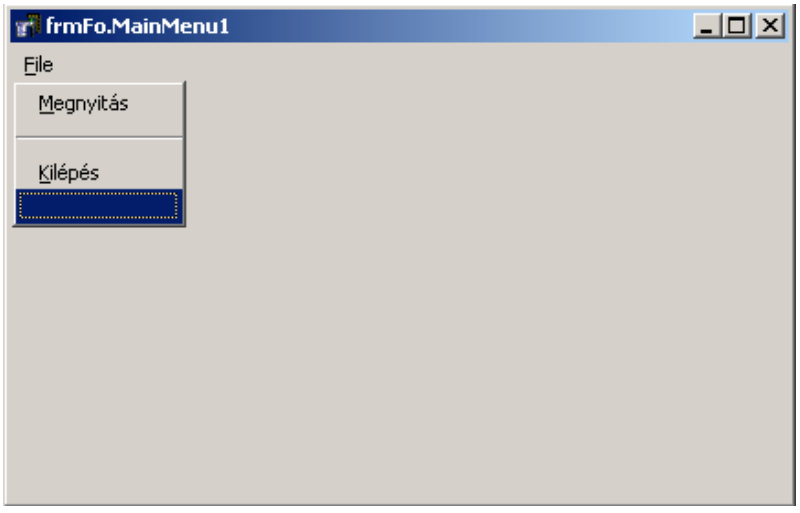
```
procedure TfrmAlap.FormClose(Sender: TObject; var Action: TCloseAction);
begin
action:=Cafree;
end;

end.
```

Elemezzük az utasításunkat. Az Action kulcsszó, egy feladat végrehajtására készteti a kódot. Maga a feladat, a CaFree utasítás. Ez a parancs, az aktuális form által lefoglalt memória területnek a felszabadítását fogja kezdeményezni, miközben a zárási (Close) művelet végrehajtásra kerül. A szimpla Close művelet (mint láttuk) nem szabadít fel memóriaterületet, mivel a főalkalmazásunk futása nem szakadt meg. Mentsünk, majd futtassuk a kódot.

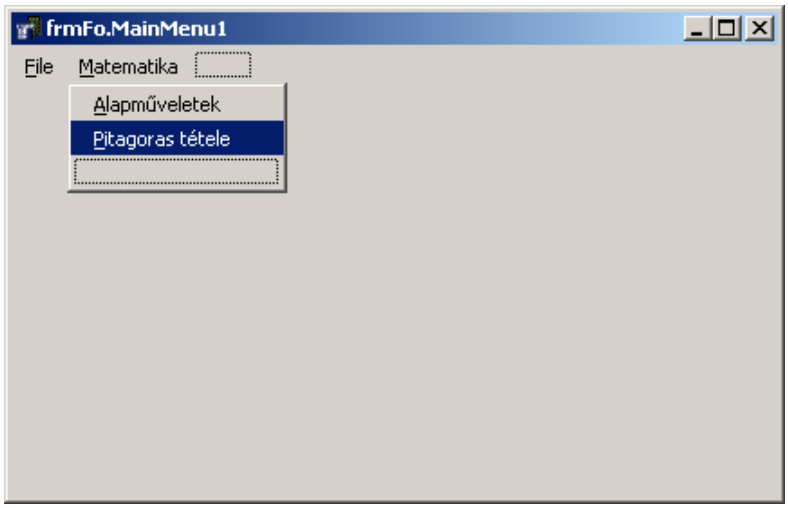
Összetett menü készítése

Látható, hogy már sikerül bezárni az ablakot, de ezután már nem tudjuk elindítani azt, hiszen programunkban nem helyeztünk el erre vonatkozóan egyetlen utasítást sem. Alapesetben (a project beállításoknak köszönhetően) minden gyermekablak megjelenik a képernyőn indításkor a főablakban, de a memória felszabadítása után már magától nem fog megjelenni.
Elvileg elhelyezhetnénk egy parancsgombot a főűrlapon, melynek eseményéhez megírjuk majd a kódot, de mivel több alkalmazást is futtatni szeretnénk, vagyis viszonylag nagy mennyiségű gombra lenne szükségünk, ezért az átláthatóság érdekében egy menüben fogjuk lehetővé tenni az alkalmazásaink indítását.

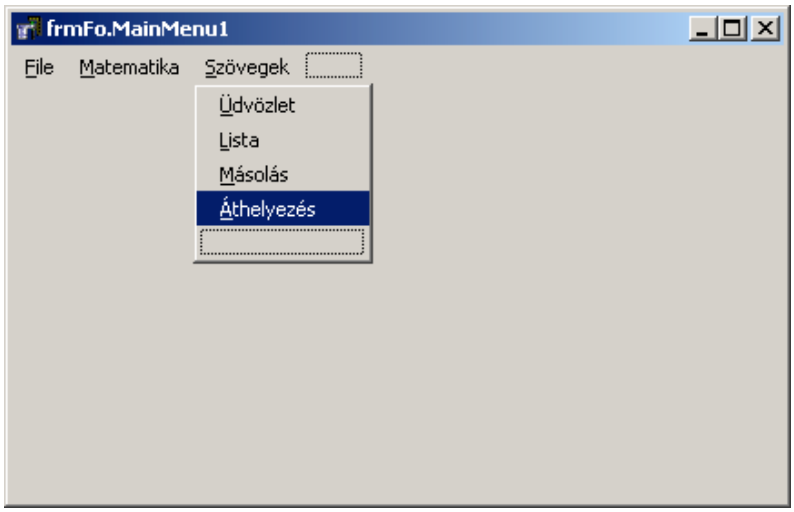
5. Lépjünk át tervező nézetben a főűrlapunkra, majd a Standard palettáról válasszuk ki a MainMenu elemet, és helyezzük el a formon. Rögtön kattintsunk duplán az elemre (nem állítunk semmit a tulajdonságain) hogy a menüszerkesztőt elindíthassuk. Legyen az első menünk a: &File. Természetesen gyorsbillentyűvel is lehetővé szeretnénk tenni a menüink indítását. Ha beírtuk, kattintsunk rá a File menünkre, majd az alatta megjelenő szürke ablakra. Itt folytassuk a menüt, a menüpontokkal: &Megnyitás, alatta legyen egy „-„ jelet tartalmazó menüpont (ezzel tudjuk tagolni a menüket), majd utolsó elemként: &Kilépés. Ha a beírás közben nem reagálna a Delphi, akkor a Caption mezőket kell feltöltenünk a fenti adatokkal:



Kattintsunk ezt követően a File menünk melletti szürke területre, majd a Caption-be írjuk be: &Matematika. Ebben a menüpontban helyezzük majd el a matematikai műveleteinket tartalmazó alkalmazásokat. Ez a menüpont tartalmazza a következő elemeket:

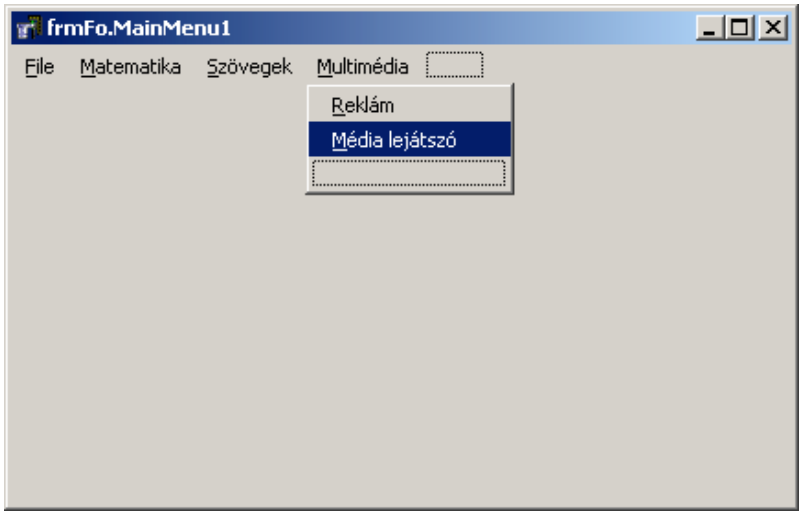


Menjünk a Matematika menü melletti szürke területre, majd írjuk be a következő menüpontot: &Szövegek. Készítsük el az almenüpontokat:



Következő menüpontunk a &Multimédia:

Utolsó menüként (a windows konvencióknak megfelelően) készítsük még el a S&úgó menüpontot a &Névjegy almenüvel. A S&úgó nem elírás, ne feledkezzünk meg róla, hogy a &Szerkesztés menühöz már felhasználtuk az „S” karaktert. Zárjuk be a menüszerkesztő ablakot, és kérjünk mentést. Futtatnunk még nem



érdemes, hisz eseményeket nem rendeltünk még a menükhöz.

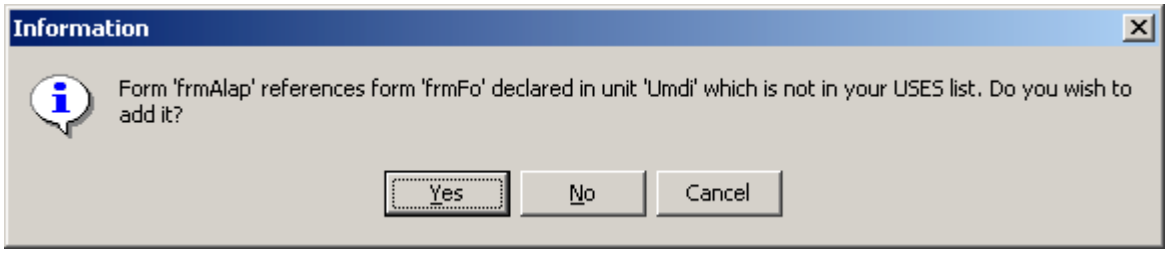
Child formok indítása

Nézzük ezután, miként indíthatóak el az alkalmazásaink. Mivel elsőként az alapműveletek Unitot adtuk a projecthez, így annak elindítását tesszük lehetővé.

6. Tervező nézetben nyissuk le a főablakunkban lévő Matematika menüpontot, és kattintsunk az alapműveletek almenüre. Rögtön meg fog nyílni a kódszerkesztő ablaka, ahol a következő kódot kell begépelnünk:

```
procedure TfrmFo.Alapmveletek1Click(Sender: TObject);
begin
frmAlap:=TfrmAlap.Create(Application);
end;

end.
```

A kódunk a form létrehozására fogja utasítani az alkalmazásunkat, vagyis az előbb bezárt ablakunkat, újra meg fogja jeleníteni a főablakban. Kérjünk mentést, majd próbáljuk futtatni az alkalmazásunkat. Ne ijedjünk meg, a Delphi egy kérdést intéz hozzánk, melyben arról érdeklődik, hogy az általunk a főablakban lévő kód, egy másik Unitban lévő formot szeretne megnyitni, amihez a USES listában szerepelnie kellene a hivatkozásnak.



A kérdés: felvegye-e ő az elemet a listába, vagy majd mi szeretnénk kézzel, esetleg hagyjon minket békén… Természetesen válaszoljunk igennel (Yes) a kérdésre, így megkíméljük magunkat egy felesleges gépeléstől. Ezt követően mentsünk ismét, majd futtassunk.

Űrlapok nyitott állapotának vizsgálata

Úgy tűnik minden rendben, ha bezárjuk a gyermekablakot, akkor a menüből újra el tudjuk indítani. Csakhogy… Ha újra felmegyünk a menübe, akkor látni fogjuk, hogy a gyermekablakunkat többször (gyakorlatilag memóriakapacitásunktól függően korlátlan számban) el tudjuk újra indítani, ami ugye nem szép, és rengeteg ablak bezárását is kérni fogja… Mit tehetünk? Nyilván meg kellene vizsgálnunk a kódban, hogy létezik-e már a form, és ha igen, akkor a menü ne tegyen semmit, egyébként pedig hozza létre azt. Módosítsuk a kódunkat a következő elágazással:

```
procedure TfrmFo.Alapmveletek1Click(Sender: TObject);
begin
If frmAlap.Active then
else
frmAlap:=TfrmAlap.Create(Application);
end;

end.
```

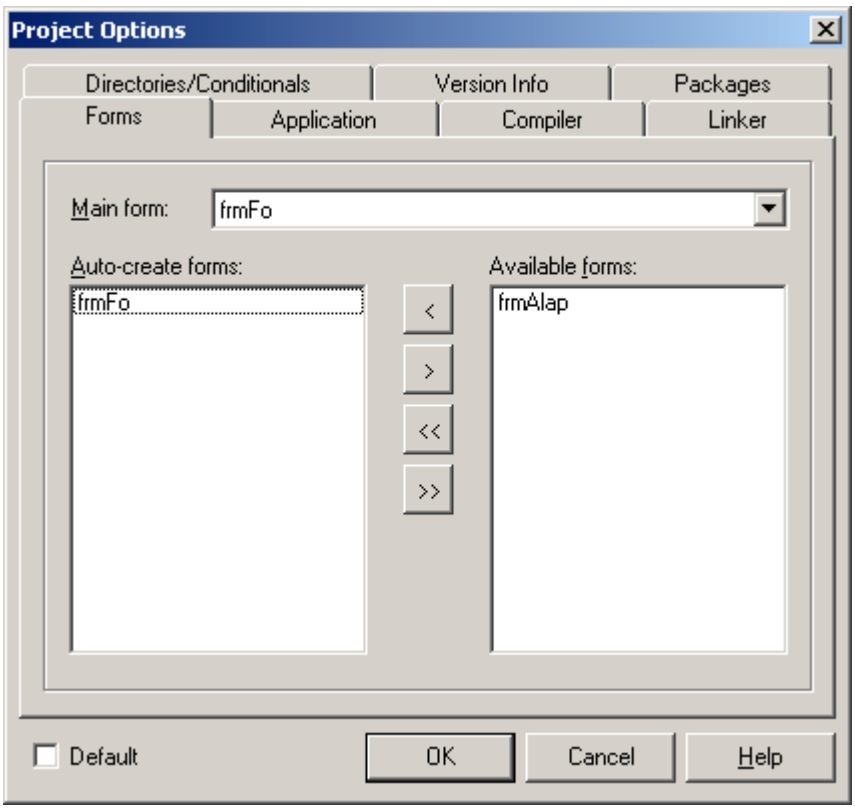
Természetesen az a szép a Delphiben, hogy nem csupán ezt a lehetőséget használhatnánk, játszhatunk például azzal a módszerrel is, hogy a gyermekablakhoz

tartozó menüpontot tesszük inaktívvá, ha megnyílt a gyermekablak, majd újra elérhetővé, ha felszabadítjuk a memória területet. És még sorolhatnánk a további lehetőségeket, melyeknek csupán fantáziánk szabhat határt.
Van azonban még egy probléma, nevezetesen az, hogy a gyermekablak a főalkalmazás indításakor automatikusan megjelenik. Ez most talán még nem tűnik zavarónak, de ne feledjük, hogy van még néhány ablakunk, amit használni fogunk. Nem lenne szép az alkalmazás, ha a felhasználónak azzal kellene kezdeni a tevékenységét, hogy a felesleges ablakokat bezárja. Mit tehetünk?

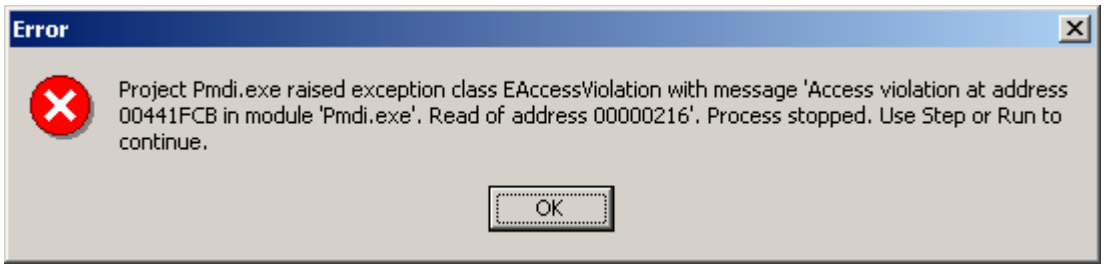
Child formok automatikus megjelenésének szabályozása

Azt, hogy egy gyermekablak megjelenjen-e a főalkalmazás indításakor, a Delphi Project menüjéből az Options almenüvel tudjuk szabályozni.

7. Hívjuk elő a Delphi Project menüből az Options almenüt. Amit itt be kell állítanunk, az a Forms panelen található. Lépjünk át oda (ha nem az lett volna az alapértelmezett), majd a megjelenő ablakban alul a bal oldalon található Auto-create forms listából dobjuk át az frmAlap-ot (akár dupla kattintással) a másik oldalra, így az nem fog megjelenni az induláskor.



Hagyjuk jóvá a tevékenységünket az OK gombra kattintva, majd mentsünk, és futtassunk. Természetesen nem kell meglepődnünk, nyilván túl egyszerűen mennek a dolgok, esedékes tehát egy újabb hibaüzenet. Minden rendben lévőnek látszik, amíg meg nem próbáljuk elindítani a meglévő gyermekűrlapunkat:



Ha tudomásul vettük a Pmdi.exe hibaüzenetét (OK), látni fogjuk, hogy a kódban a feltételvizsgálatnál akadt ki a gép, melyet az űrlapunk menüből történő indítására írtunk. A probléma most az, hogy mivel nem jött létre automatikusan a formunk, így nem tudjuk annak Active tulajdonságát sem vizsgálni. Természetesen, ha kivennénk az elágazást, akkor működne a menü, de újra szembesülnénk a többször megnyitható ablakok problémájával… Mit tegyünk tehát?

Child formok eltüntetése

A probléma abból fakad, hogy egy MDI alkalmazás gyermek ablakainak nem állíthatjuk át sem a Visible, sem a Hide tulajdonságát, sem közvetlenül a tulajdonságlapon (ekkor rögtön hibaüzenetet kapunk a Delphitől), sem a kódból. Ekkor ugyan beírhatjuk az utasítást, de a futtatás során fog kiderülni a fenti állítás igaz volta. Természetesen ki lehet próbálni a fentieket, a jegyzet ettől el fog tekinteni. A legegyszerűbben talán úgy oldhatjuk meg a fenti problémát, ha létrehozzuk kódból az űrlapunkat, majd szintén a kódban be is zárjuk azt. Ettől kezdve már vizsgálni tudjuk annak tulajdonságait, és mégsem jelenik majd meg a képernyőn.

8. Válasszuk ki tervező nézetben a főablakunkat (frmFo), majd az Events lapon keressük meg a legelső OnActivate eseményt. A mezőben duplát kattintva hívjuk meg a kódszerkesztőt, és írjuk be a következő kódot:

```
procedure TfrmFo.FormActivate(Sender: TObject);
begin
frmAlap:=TfrmAlap.create(Application);
frmAlap.Close;
end;

end.
```

Mivel az űrlapunk nem jött létre a főűrlap indulásakor automatikusan, így mi hozzuk azt létre az első utasítással, miután aktívvá vált a főablak (OnActivate). Ezt követően azonban rögtön be is zárjuk, a Close metódussal, ahol korábban felszabadítottuk az általa lefoglalt memória területet.

Tulajdonképpen elkészült az első korábban készített alkalmazásunk végrehajtását lehetővé tévő keretalkalmazás. Annyit még tehetünk (alkalmazásunk csinosítása érdekében), hogy a gyermekablakot a főablakon belül teljes képernyőn jelenítjük meg. Ez ugyan a gyermekablak elrendezését nem változtatja meg (sőt egy kissé féloldalassá válik az alapműveleteket végző alkalmazásunk), de legalább már nem nyitja meg a gyermekablakot a belső ablakban össze-vissza a főablak.

9. Keressük meg az alapműveleteket végző formunk (frmAlap) tulajdonságlapján a WindowState tulajdonságot, és állítsuk wsMaximized-re.

Mentsünk, majd teszteljünk, győződjünk meg róla, hogy minden rendben működik. Ha megfelelőnek találtunk mindent, akkor nincs más hátra, mint valamennyi eddigi

munkánkat próbáljuk meg önállóan elhelyezni a keretalkalmazásban, hozzájuk rendelve a neki megfelelő menüpontot:

File: Megnyitás (frmOpen)
Kilépés (Egyszerű Close művelet megerősítéssel)
Matematika: Alapműveletek (már elkészítettük)
Pitagoras tétele (frmPita)
Szövegek: Üdvözlet (frmHello)
Lista (frmLista)
Másolás (frmMasol)
Áthelyezés (frmhelyez)
Multimédia Reklám (frmReklam)
Média lejátszó (frmVideo)
Súgó Névjegy (frmNevjegy)

Nézzük ismétlésként, (összefoglalásként) lépésről lépésre mit is kell tennünk:

1. A Unit beillesztése a projectbe (Project/Add To Project)
2. A beillesztett unitban lévő form megjelenítése (View/Forms új form kiválasztása)
3. A gyermekablak FormStyle tulajdonságának átállítása MDIChild-ra.
4. A gyermekablak OnClose eseményének megírása a memóriaterület felszabadítása (action:=CaFree;)
5. A főablakunk megfelelő menüpontjának megírása az elágazással.
6. Mentés után futtatáskor a USES list bővítésének engedélyezése (OK)
7. A gyermekűrlap automatikus megjelenésének kizárása a főprogramunk indulásakor (Project/Options/Forms Auto-create forms oldalról áthelyezni)
8. A főablakunk OnActivate eseményét kibővíteni az új űrlapunk megnyitásának, majd azonnali bezárásának utasításával.

Összességében elmondható, hogy a moduláris fejlesztések egyik példáját láthatjuk elkészült alkalmazásunkban. Minden projectünk önállóan is működik, egymásba ágyazásuk pedig már talán nem tekinthető komoly kihívásnak ezek után.
Az egyetlen összetettebb kódot a főablakunk OnActivate eseménye tartalmazza:

Sajnos csak így tudtuk megkerülni a létrejövő formok azonnali megjelenését a főablakban. Ez természetesen nem tökéletes megoldás (sőt), hiszen jelen esetben akár szüksége van a felhasználónak az összes funkcióra, akár nincs, azok be fognak töltődni, vagyis a programunkra még ráfér egy erős optimalizálás. Ez különösen zavaró, hisz lassabb gépen (de a mieinken is) látszik, amint létrejönnek, majd eltűnnek a formok. Ezt kiküszöbölhetjük (a korábban már említett) menüpontok kattintásra történő inaktívvá tételével, majd (az adott form bezárásakor, a memóriaterület felszabadításakor) annak újra aktívvá tételével.
Természetesen ez megint azzal fog járni, hogy a kódunkat erősen módosítani fogjuk. Például rögtön töröljük ki azt a bizonyos zavaró űrlapmegnyitás-eltüntetés sorozatot:

1. Keressük meg a főűrlapunk tulajdonságlapján lévő OnActivate eseményt, majd kattintsunk rá duplán. A megjelenő ablakban töröljük most a korábban nagy kínnal beírt valamennyi utasításunkat, csak a Begin End páros maradjon. Ha mentést kérünk, látni fogjuk, hogy a Delphi automatikusan eltávolítja kódunk hivatkozásait (például az OnActivate panel az objektum-kezelőben üressé vált).

Menüpontok engedélyezése, tiltása

2. Ha azt szeretnénk, hogy a menüpontra kattintva megjelenjen a kívánt űrlap, és közben a menü inaktívvá váljon, akkor nincs szükségünk a felesleges elágazásra sem, hiszen a menü aktív-inaktív volta tájékoztat a form állapotáról is. Lépjünk tehát a File menünk Megnyitás almenüjéhez, és kattintsunk rá. A megjelenő kódot módosítsuk a következő két sorra:

```
procedure TfrmFo.Megnyits1Click(Sender: TObject);
begin
frmOpen:=TfrmOpen.Create(Application);
frmFo.Megnyits1.enabled:Enabled=False;
end.
```

Az első sorban az űrlap megnyitására fogjuk utasítani a programot, majd a másodikban a jelenlegi menüpont engedélyezését átállítjuk False-ra. Így elindul az űrlap, és a felhasználónak nem lesz lehetősége még egyszer elindítani azt. Ha megfigyeljük a Delphi konvenciónak megfelelő nevet láthatjuk a kódban, amit persze nem kell fejből tudnunk, elég ha az eljárásunk nevét megnézzük.
Minden rendben, nem érkezik hibaüzenet, de a gyermekablak bezárása után is inaktív marad a menüpont, vagyis a felhasználó csak egyszer tudja elindítani a Child-ot. Tegyünk valamit ennek kiküszöbölése érdekében:

3. Keressük meg a gyermekablakot (frmOpen), majd az események közül keresük ki az OnClose (bezárásra) eseményt. Itt csupán egy újabb sort kell hozzáfűznünk a kódhoz, mégpedig az előzőleg letiltott menü engedélyezését. Ha nem másoltuk ki a kódot, vagy nem tudjuk elolvasni innen a jegyzetből az előző utasításainkat, akkor a Unitokban a fülecskék között váltva kimásolhatjuk előző utasításunkat, melynek csak a végét kell átírnunk: True-ra: frmFo.Megnyits1.Enabled:=True;

Ha mentünk, és futtatni szeretnénk, egy már ismerős kérdést kapunk majd a géptől. Előzőleg a főablakból hivatkoztunk a gyermekablak Unit részére, most viszont a gyermek ablak Unitjából hivatkozunk a főablakunk Unit részére. Ezt a hivatkozást is hagyjuk jóvá a YES gombbal.

Teszteljünk ismét, majd ha mindent rendben lévőnek találtunk, írjuk át a kódokat ennek megfelelően valamennyi gyermekablakunkban.

Ellenőrző kérdések

I.

KÉREM VÁLASZOLJON A FELTETT KÉRDÉSEKRE!

1. Mire használjuk az Action:=CaFree utasítást?
2. Mi a szerepe az MDI formnak?
3. Miért jelennek meg a gyermekablakok indításkor?
4. Mire használható az OnActivate esemény?
5. Adja meg az MDI formok definícióját és típusait!

II.

KÉREM OLDJA MEG A KÖVETKEZŐ FELADAT(OKA)T!

1. Hozzon létre egy menüvezérelt MDI alkalmazást mely a File menüben a Megnyitás, Nyomtatás, Kilépés lehetőségekkel rendelkezik. A megnyitást korábban elkészített alkalmazásának (OPEN) felhasználásával építse be, a nyomtatás indítsa el a nyomtatók párbeszédpanelt (beépítheti a főablakba!), a kilépés pedig megerősítésre bezárja az alkalmazást! A munkát mentse a Delphi könyvtárában létrehozandó FELADAT mappába Usajat, és Psajat neveken!

Delphi tippek, és trükkök.

Bevezetés

Az eddig tanultak adtak némi alapot a Delphit most kezdők számára, az alkalmazások használatához. Ebben a fejezetben egyszerű alkalmazásokon belül megismerhetünk néhány kódot, melyeknek segítségével konkrét, ritkán használt, de néha nagyon hasznos műveleteket végezhetünk el a Delphiben.

Analóg óra megjelenítése a formon:

Bár a Delphi komponenseiből építhetőek fel a legösszetettebb alkalmazások, mint látni fogjuk ebben a feladatban, a kód is rejt néhány érdekességet magában.
Hozzunk létre egy ORA nevű könyvtárat a Delphin belül, majd indítsuk el a Delphit.

1. A megjelenő form Name: frmOra, caption: Analóg óra a formon. A ClientWidth: 400, a ClientHeight: 400
2. Save All: Uora, Pora
3. Helyezzünk el egy Timer-t a formon, majd kattintsunk az időzítőre eseményhez, ahová írjuk be ezt a viszonylag összetett kódot:

```
procedure TfrmOra.Timer1Timer(Sender: TObject);
const aspect:real=1.0;
var
  Form1: TForm1;
  xc,yc,r,i,d: integer;
  Present: TDateTime;
  Hour, Min, Sec, MSec: Word;
begin
    xc:=clientwidth div 2;
    yc:=clientheight div 2;
    r :=(xc+yc) div 4;
    d:=(xc+yc) div 64;
    Canvas.Pen.Color := clWhite;
    canvas.ellipse(xc-r,yc-r,xc+r,yc+r);
    Canvas.Pen.Color := clBlack;
    for i:=1 to 12 do
        begin
        canvas.moveto(round(xc+(r-4*d)*cos(i*pi/6)),round(yc+(r-4*d)*sin(i*pi/6)));
        canvas.lineto(round(xc+(r-d)*cos(i*pi/6)),round(yc+(r-d)*sin(i*pi/6)));
        end;
    for i:=1 to 60 do
        begin
        canvas.moveto(round(xc+(r-2*d)*cos(i*(pi/30))),round(yc+(r-2*d)*sin(i*(pi/30))));
        canvas.lineto(round(xc+(r-d)*cos(i*(pi/30))),round(yc+(r-d)*sin(i*(pi/30))));
        end;
    Present:= Now;
    DecodeTime(Present, Hour, Min, Sec, MSec);
    canvas.pen.Width:=3;
    i:=Min-15;
    canvas.moveto(round(xc+(-d)*cos(i*(pi/30))),round(yc+(-d)*sin(i*(pi/30))));
    canvas.lineto(round(xc+(r-5*d)*cos(i*pi/30)),round(yc+(r-5*d)*sin(i*(pi/30))));
    canvas.pen.Width:=5;
    i:=(Hour - 12) * 5 + (round(Min / 12))-15;
    canvas.moveto(round(xc+(-d)*cos(i*(pi/30))),round(yc+(-d)*sin(i*(pi/30))));
    canvas.lineto(round(xc+(r-10*d)*cos(i*(pi/30))),round(yc+(r-10*d)*sin(i*(pi/30))));
    canvas.pen.Width:=1;
    i:=Sec-15;
    canvas.moveto(round(xc+(-d)*cos(i*(pi/30))),round(yc+(-d)*sin(i*(pi/30))));
    canvas.lineto(round(xc+(r-5*d)*cos(i*(pi/30))),round(yc+(r-5*d)*sin(i*(pi/30))));
end;

end.
```

Mentsünk, majd futtassunk. Látni fogjuk, hogy a kód eredményeképpen űrlapunkon kirajzolódik az analóg óra, az aktuális időpontot mutatva. A kód részletes

magyarázatát mellőznénk, a Pascal-ban volt már szó az itt szereplő függvényekről, és átalakításról. Figyelmet érdemelhet esetlegesen a konstansunk, és a helyi változóink, melyeket a begin előtt definiáltunk. Amit érdemes észrevennünk futtatás közben (tesztelés, hibajavítás), hogy a statikus formra rajzolás eredményeképpen, az űrlap méretének változtatására az eredetileg kirajzolt objektumunk mellett a növelt (vagy csökkentett) mérethez igazodó óra is kirajzolódik. Ezt a problémát korrigálhatjuk a form méretének statikussá tételével.

Egy összetett példaalkalmazás (TIPP):

Start menü indítása Delphiből:

A következő alkalmazásunk egy viszonylag kaotikus munka lesz, rendszertelenül adunk néhány gombot a formunkra, melyek különböző érdekes műveleteket hajtanak majd végre.
Hozzuk létre a project tárolásához szükséges TIPP alkönyvtárat a Delphiben.

1. Indítsuk el a Delphit, majd a form tulajdonságait módosítsuk: Name:frmTipp, Caption: Tippek, és trükkök Delphiben.
2. Kérjünk mentést: Utipp, Ptipp
3. Helyezzünk el egy Button-t a formon (Standard komponens), majd a neve legyen: btnStartmenu, a Caption: Start menü indítása.
4. Kattintsunk duplán a gombon, és a kódszerkesztőbe írjuk be a következő sort:

```
procedure TfrmTipp.btnStartmenuClick(Sender: TObject);
begin
Sendmessage(Application.Handle,WM_SYSCOMMAND,SC_TASKLIST,0);
end;
```

Ha rákattintunk a gombunkra, tulajdonképpen szimulálni fogjuk a start menüre történő kattintást, vagyis a kód elindítja a Windows start menüjét.

Dos parancssor indítása Delphiből:

Bizonyos esetekben szükségünk lehet a parancssori ablak megjelenítésére. Ekkor jöhet jól a következő gomb, vagy utasítás:

1. Hozzunk létre a formon egy másik gombot, tulajdonságai: Name: btnDos, Caption: Dos program.
2. Kattintsunk rá duplát, majd a megjelenő ablakba írjuk a következő kódot:

```
procedure TfrmTipp.btnDosClick(Sender: TObject);
begin
WinExec('command.com',sw_ShowNormal);
end;
```

Kódunk, a Delphiből elindítja a parancssort, és átadja neki a vezérlést. Maga a WinExec függvény, a windows alkalmazásainak elindítását teszi lehetővé. Természetesen lehetőségünk van paraméterezve elindítani a parancssort, például a progman.exe indításához a parancssorban a következő módon kell megváltoztatnunk a fenti utasítást:

```
procedure TfrmTipp.btnDosClick(Sender: TObject);
begin
WinExec('command.com /c progman.exe',sw_ShowNormal);
end;
```

A függvény második paramétere (sw_ShowNormal) az ablak megjelenítés módját szabályozza. Ha nem szeretnénk, hogy a felhasználó lássa az elindított alkalmazást, akkor használhatjuk az sw_Hide paramétert.
Nyilvánvaló, hogy bármilyen Dos alatti alkalmazást elindíthatunk a parancsértelmezőt paraméterezve a fenti módszerrel. De hogyan indíthatóak a windows alkalmazásai?

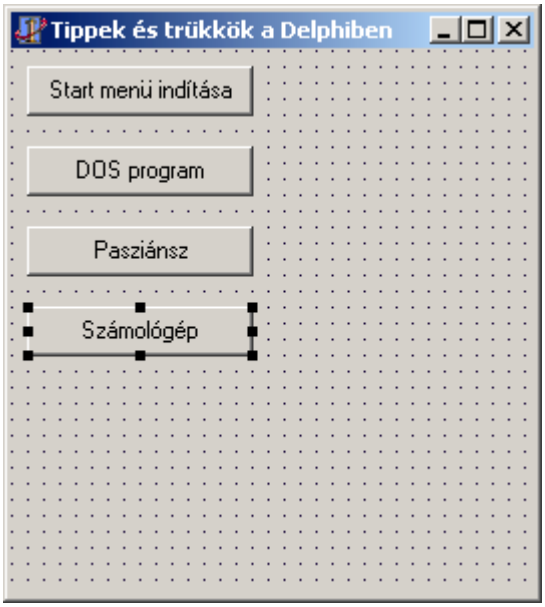
Windows alkalmazások indítása Delphiből:

Ahogy az előzőekben láttuk, a parancssor elindításához a WinExec függvény állt rendelkezésünkre. Mint a nevéből is látható, természetesen a parancssort is a Windows operációs rendszer alól indítottuk el, így a Windows beépített alkalmazásai szintén ennek a függvénynek a paraméterezésével indíthatóak el:

1. Hozzunk létre egy újabb gombot a formon, Name: btnFreecell, Caption: Pasziánsz.
2. Kattintsunk duplát a gombra, majd a kódszerkesztőbe írjuk be a lenti kódot:

```
procedure TfrmTipp.btnFreecellClick(Sender: TObject);
begin
WinExec('freecell.exe',sw_ShowNormal);
end;
```

Ha futtatunk, látni fogjuk, hogy minden további nélkül elindul az alkalmazás.
A számlázó, és készletnyilvántartó programokban néha lehetőségünk van a számla kiállításakor egy gomb segítségével elindítani a számológépet. Ekkor csupán az alkalmazás nevét kell ismernünk, és annak megfelelően módosítani a fenti kódsort.
Készítsük el önállóan a számológépet elindító gombot a formon: (Name:btnCalc)



Mentsünk, teszteljünk, majd ha mindent megfelelőnek találtunk, folytassuk tovább a gombok létrehozását.
Észre kell vennünk, hogy a WinExec függvény, csak a Windows közvetlenül (a rendszerkönyvtárban található) alkalmazásainak indítását teszi lehetővé. Abban az esetben, ha például a böngészőt szeretnénk elindítani, már nem elegendő a WinExec. Mit tegyünk ilyenkor?

A böngésző indítása:

A rendszerkönyvtáron kívül elhelyezkedő alkalmazások indításához használhatjuk a ShellExecute függvényt. Csakhogy a függvény (amennyiben meg szeretnénk hívni) nem az alapértelmezett Unit-ok között található, így szükségünk lesz a USES lista módosítására is. A ShellExecute, a ShellApi unit része, így használatához ezt a unitot fel kell vennünk a USES részbe.

1. Helyezzünk el egy gombot a formon, Name: btnBongeszo, Caption: Böngésző.
2. Kattintsunk duplán a gombra, majd a kódszerkesztőbe írjuk be a ShellExecute függvényünket a következő módon paraméterezve:

```
procedure TfrmTipp.btnBongeszoClick(Sender: TObject);
begin
ShellExecute(Handle, 'open', 'http://www.index.hu', '', '', SW_SHOWDEFAULT)
end;
```

Ha a mentés után megpróbálunk futtatni, a kód a függvényünknél hibát fog jelezni, hisz a ShellApi még nem része a USES listának.

3. A kódszerkesztőben a gördítősáv segítségével keressük meg a kód felső részén elhelyezkedő USES kulcsszót, majd az ott található Unit listát egészítsük ki a ShellApi-val:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ShellApi;
```

A mentés, és futtatás eredményeként elindul az alapértelmezett böngésző, az általunk megadott paraméternek megfelelően.
Amennyiben az alapértelmezett levelezést szeretnénk lehetővé tenni a felhasználónak a programunkon belül, akkor szintén ezt a függvényt használhatjuk, természetesen már nem kell a ShellApi-t újra felvennünk, hisz már megtettük azt.

Levelezés a programunkból:

1. Hozzunk létre egy újabb gombot a formon: Name: btnMail, Caption: Levelezés
2. Kattintsunk duplán a gombunkra, majd jöhet a kód:

```
procedure TfrmTipp.btnMailClick(Sender: TObject);
begin
ShellExecute(Handle, 'open', 'mailto:tibusoft@freemail.hu', '', '', SW_SHOWDEFAULT);
end;
```

Az előző függvényünk csupán a HTML mailto klauzájával módosult. Természetesen ezt még tovább lehet bővíteni, az ott is megszokott módokon. Ha például szeretnénk kitölteni a felhasználó helyett a Subject részt, akkor a kódot a következőre kell módosítanunk:

```
procedure TfrmTipp.btnMailClick(Sender: TObject);
begin
ShellExecute(Handle, 'open', 'mailto:tibusoft@freemail.hu?subject=Kérdésem lenne', '', '', SW_SHOWDEFAULT);
end;
```

A CD meghajtó ajtajának nyitása, zárása:

Az előző példában láthattuk, hogy a Unitokban rengeteg segédfüggvény elérhető, melyeket különböző feladatok végrehajtásához igénybe vehetünk. Következő példánkban (még mindig a Tipp feladatsorban) egy hardvert kezelő eljárást fogunk kezdeményezni.

1. Helyezzünk el egy GroupBox-ot (Standard paletta) a formunk jobb felső részén. Tulajdonságai: Name: grpCD, Caption: CD meghajtó vezérlése.
2. Helyezzünk rá a GroupBox-ra egy Buttont. Tulajdonságai: Name: btnNyit, Caption: Tálca nyitása.
3. Kattintsunk duplán a gombon, majd a kódszerkesztőbe írjuk be a kódot:

```
procedure TfrmTipp.btnNyitClick(Sender: TObject);
begin
mciSendString('Set cdaudio door open', nil, 0, 0);
end;
```

Ha megpróbálunk mentés után futtatni, hibaüzenetet fogunk kapni, az általunk meghívott függvény ugyanis nem szerepel a deklarációs rész Unitjaiban. Az mciSendString függvény az MMSYSTEM Unitban található, így fel kell azt vennünk a USES deklarációs részbe:

4. A kódszerkesztőben keressük meg a felső részben található USES részt, és egészítsük ki a kódunkat az MMSYSTEM unitra történő hivatkozással:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ShellApi, MMSystem;
```

Ezt követően mentés után már futtathatunk, s ha mindent megfelelően hajtottunk végre, a gombunkra történő kattintás (és némi töprengő várakozás után, különösen ha több CD/DVD meghajtó is található a gépben) ki fog nyílni az alapértelmezett CD meghajtó tálcája. Természetesen kezdeményezhetjük annak bezárását is, egy másik gomb segítségével:

5. Az előzőleg elhelyezett GroupBoxba, az előző gomb alá tegyük fel a következő Button-t. Name: btnZar, Caption: Tálca bezárása.
6. Kattintsunk a gombunkra duplán, és írhatjuk a kódot:

```
procedure TfrmTipp.btnZarClick(Sender: TObject);
begin
mciSendString('Set cdaudio door closed', nil, 0, 0);
end;
```

Mint látható, a kód csupán a „closed” kifejezésben tér el az előzőleg megírt kódtól, így annak másolását is kezdeményezhetjük.

Ismét utalnánk arra, hogy a különböző Unit-ok, rengeteg belső függvényt tartalmaznak, melyeket feladatainkhoz igénybe vehetünk. Mindezek részletes ismertetése túl nagy feladat lenne, és messze túlmutatna e jegyzet terjedelmein. Mivel ez nem is célunk, csupán az alapok megismertetése, így az esetleges további függvények listájához kérjük az angol nyelvű súgót az F1 billentyűvel. Ha kijelölünk egy függvényt a kódban, és így kérjük a súgót, akkor amennyiben szerepel a függvény a súgóban, akkor rögtön az adott témát fogjuk kapni.

Futó alkalmazás ablakának elrejtése, visszahívása:

Tételezzük fel, hogy az iskolában, munkahelyen nem megengedetten Internetezünk, és állandó problémát jelent, hogy a főnök (tanár) megjelenésekor be kell zárnunk a böngészőt, mert a tálcán észreveheti azt. Ezzel az addigi levelünk (mit keservesen begépeltünk) elszáll, a játék állása elveszik, a csevegés a legizgalmasabb pillanatban hirtelen megszakad, és kezdhetjük újra a folyamatot, mely a következőkben várhatóan ismét hasonlóan meg fog szakadni. A megoldás, a böngésző gombnyomásra történő elrejtése, majd megjelenítése:

1. Helyezzünk a formunkra az előző GroupBox alá egy újabb GroupBox-ot. Tulajdonságai: Name: grpNet, Caption: Megnyitott IE böngésző vezérlése.
2. Helyezzünk egy gombot az új groupBox-ba: Name: btnRejt, Caption: Böngésző elrejtése.
3. Kattintsunk új gombunkra, és a kódba írjuk be a következőket:

```
procedure TfrmTipp.btnRejtClick(Sender: TObject);
var taskbar:HWND;
begin
taskbar:=FindWindow('IEFrame',nil);
ShowWindow(taskbar,SW_HIDE);
end;
```

Mint látható, definiálnunk kell egy taskbar nevű változót, majd értéket adva neki, a ShowWindow függvénnyel tudjuk eltüntetni a keresett alkalmazást. Természetesen ennek van egy alapfeltétele: a böngészőnek futnia kell. További hiányossága a kódunknak, hogy amennyiben más böngészőt használunk, arra jelenlegi kódunkkal nem fogunk tudni „nyomást” gyakorolni. Ellenben az Internet Explorer, a gombunk megnyomásakor el fog tűnni a képernyőről, mégpedig úgy, hogy a tálcán sem lesz látható. Persze illene visszahozni:

4. Helyezzük el az előző gombunk alá a következő gombot: Name: btnMutat, Caption: Böngésző felfedése.
5. Duplát kattintva új gombunkon, írjuk a kódszerkesztőbe a következőt:

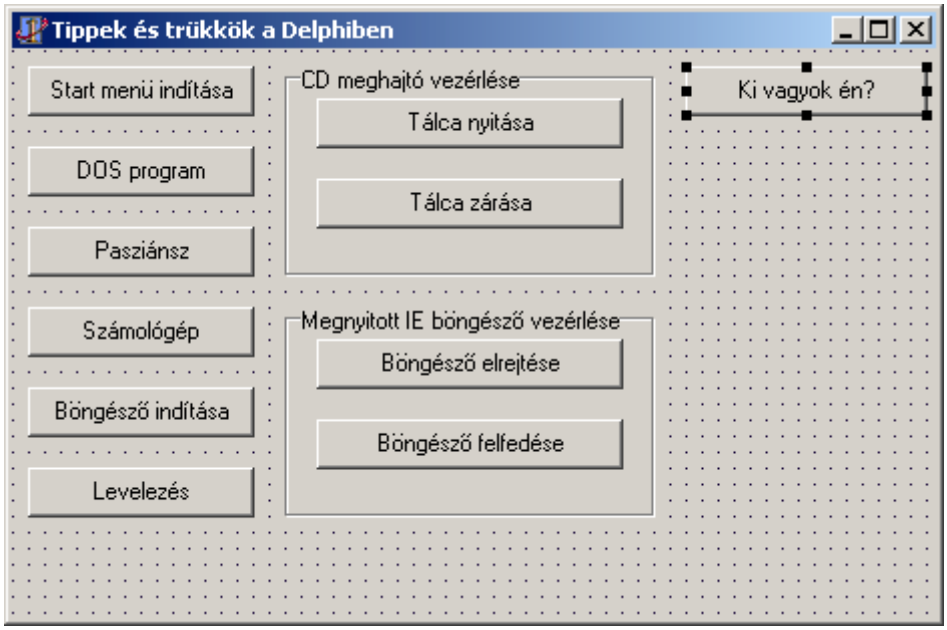
```
procedure TfrmTipp.btnFelfedClick(Sender:TObject);
var taskbar:HWND;
begin
taskbar:=FindWindow('IEFrame',nil);
ShowWindow(taskbar,SW_RESTORE);
end;
```

Vegyük észre, hogy már a gomb nevében utalunk rá: itt nem az alkalmazás indításáról van szó, csupán annak visszaállításáról, megjelenítéséről, előtérbe hozásáról. A kódunk itt is csupán csak az SW_HIDE (elrejtés), és SW_RESTORE (visszaállítás) kifejezésekben tér el egymástól.

Ki az aktuális felhasználó?

Az operációs rendszer rengeteg információt tárol, és tart nyilván a felhasználókkal, a programokkal, és az éppen futó folyamatokkal kapcsolatosan. Ezeket az adatokat mi is elérhetjük, ha hozzá tudunk férni azokhoz az információkhoz, hogy milyen helyekre kerültek a bennünket érdeklő adatok. Következő példánkban az operációs rendszerbe bejelentkezett aktuális felhasználót fogjuk megjeleníteni egy gomb segítségével.

1. Hozzunk létre egy újabb gombot, ezt a GroupBox-ok mellett jobbra egy új oszlopban legfelülre helyezzük el. Name: btnFelhaszn, Caption :Ki vagyok én?



2. Duplát kattintva a gombon, írjuk be a következő kódot:

```
procedure TfrmTipp.btnFelhasznClick(Sender: TObject);
var
NameBuf: array[0..80] of Char;
SizeBuf: LongWord;
begin
SizeBuf := Sizeof(NameBuf);
GetUserName(NameBuf, SizeBuf);
Showmessage(NameBuf);
end;
```

Ha mentünk és futtatunk, a gombunk megnyomásának hatására, a tárolt érték egy üzenetablakban megjelenik.

A futó alkalmazás elérési útja:

Néha szükségünk lehet arra, hogy kiderítsük, az aktuálisan futó alkalmazás honnan (milyen útvonalról) érhető el. Például, ha az INI állományának adatait szeretnénk elérni, vagy néhány állomány meglétét ellenőrizni. Ekkor jöhet jól a következő kódrészlet:

1. Helyezzünk el egy gombot az előző gomb alatt. Name: btnHolvan, Caption: Honnan indítottuk?
2. Kattintsunk rá duplán, majd írhatjuk a következő kódot:

```
procedure TfrmTipp.btnHolvanClick(Sender: TObject);
var
myPath : String;
begin
myPath := ExtractFilePath( Application.ExeName );
showmessage(mypath);
end;
```

A megjelenő ablakban az éppen futó alkalmazásunk teljes elérési útvonalát fogjuk látni. Ez sokszor elég is, de néha azt sem árt tudnunk, hogy hol a Windows könyvtár?

A windows könyvtár elérése:

Ebben a részben az operációs rendszer könyvtárának elérési útját, helyét vizsgáljuk meg. A kódot kiegészítjük most egy saját függvénnyel is, melyet használni fogunk.

1. Vegyünk fel egy újabb gombot, az előző alatt, Name: btnHolvan, Caption: Hol a Windows?
2. Kattintsunk duplán a gombra, majd hogy megzavarjuk majd magunkat a későbbiekben, a megjelenő ablakba írjuk be a következőt:

```
procedure TfrmTipp.btnWinClick(Sender: TObject);
begin
showmessage(Hol_a_windows());
end;
```

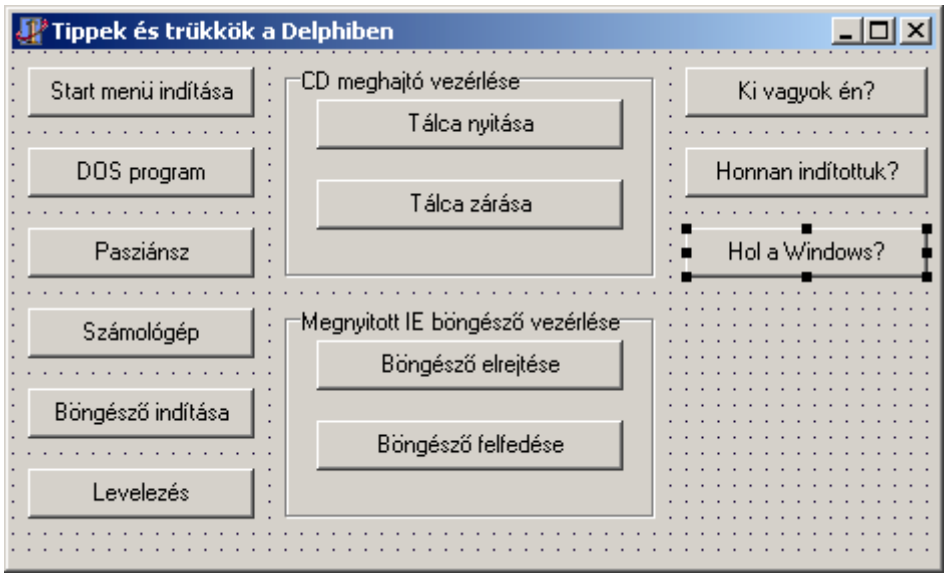
Természetesen a Hol_a_windows egy függvény, melyet még nem deklaráltunk. Nagyon meglepő lenne, ha ennek ellenére a mentés után futtatni tudnánk a programot. Nézzük a függvény írását:

Saját függvény a programon belül:

3. Keressük meg a kódszerkesztőben a görgetősáv segítségével a típusdeklarációk (Type) után található megvalósítási (IMPLEMENTATION) részt, és a procedure-k előtt definiáljuk saját függvényünket. A képen igyekeztünk jelezni a pontos helyet is, ahová írniuk kell majd a függvényt. Természetesen a Function kulcsszóval kezdődik a gépelendő szakasz, az alábbi módon:

```
    procedure btnWinClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmTipp: TfrmTipp;

implementation

{$R *.DFM}

function Hol_a_windows: string;
const
NevHossza: DWORD = 255; //a könyvtárnévhez elvileg 64 is bőven elég...
var
KonyvtNev: PChar;
begin
GetMem(KonyvtNev, NevHossza);
GetWindowsDirectory(KonyvtNev, NevHossza);
Result := string(KonyvtNev);
FreeMem(KonyvtNev, NevHossza);
end;
```

Tulajdonképpen mindegy hová kerül az általunk definiált függvény, de még azelőtt szerepeltessük, mielőtt meghívnánk azt. Míg az eljárások esetében azokat előzetesen is deklarálnunk kell, addig a függvények ezt nem kívánják meg.
Kérjünk mentést, futtassunk, majd nézzük meg minden rendben van-e?
Formunk jelenlegi állapotában már csak 3 gomb felvitelére alkalmas, helyezzük még el azokat néhány kóddal:



Hangkártya meglétének vizsgálata:

Az előzőekben volt egy hardveres eszköz (CD) hozzáférésére példa, valamint legutóbb egy saját függvény, most ötvözzük a kettőt. Példánkban azt vizsgáljuk, hogy a gép rendelkezik-e hangkártyával?

1. Következő gombunk: Name:btnHang, Caption: Van hangkártya?
2. A kódszerkesztőben pedig kezdetnek egy egyszerű utasítás:

```
procedure TfrmTipp.btnHangClick(Sender: TObject);
begin
If Van_hang then
showmessage('Hangkártya installálva!')
else
showmessage('Nincs hang!');
end;
```

Egyszerű elágazásunkban csupán a még nem létező függvényünk visszatérési értékét vizsgáljuk, így még nem fog futni az alkalmazás. Írjuk meg a függvényt:

3. Keressük meg a korábban megírt Hol_a_windows függvényt, majd deklaráljuk alá a következő saját függvényünket:

```
function Van_Hang:boolean;
begin
Result:=(waveOutGetNumDevs>0); //ez is használja a MMSystem Unitot!
end;
```

Ha korábban nem vettük fel volna a USES listába, akkor most itt akadt volna ki a kód, ugyanis a függvényben szereplő utasítás, a kommentben is szereplő MMSystem unitban érhető csak el.

Ha mentünk, és futtatunk, a párbeszédablak ki fogja elégíteni kíváncsiságunkat. Természetesen adódik rögtön egy probléma: nem tudjuk kipróbálni mindkét ágat, vagyis a teszthez el kellene távolítanunk az eszközt. Ha egy mód van rá, bízzunk magunkban, meg a megírt eljárásunkban, és hagyjuk a gép eszközeinek letiltogatását… Lássunk egy utolsó eszközt, a monitort:

Képernyő felbontásának állítása:

A következő példánkban még tovább lépünk, újabb saját függvénnyel dolgozva.

Egy gomb segítségével próbáljuk majd a felhasználónak lehetővé tenni a képernyő felbontásának megváltoztatását.

1. Helyezzük el a következő gombunkat, Name: btnFelbontas, Caption: Képernyő felbontása.
2. A kódszerkesztőbe írjuk be ezt az egyszerű elágazást:

```
procedure TfrmTipp.btnFelbontasClick(Sender: TObject);
begin
If messagedlg('Kisfelbontást szeretne?',mtconfirmation,[mbYes, mbNo],0)=mrYes ther
        kepfelbontas(800,600,32)
else
        kepfelbontas(1024,768,32);
end;
```

3. Keressük meg a kódszerkesztőben a korábbi függvényeinket, majd helyezzünk el egy újabbat az előzőek után:

```
function KepFelbontas(WResolution, HResolution, Depth: DWORD) : Boolean;
var
i: Integer;
DevMode: TDevMode;
begin
Result := False;
i:=0;
while EnumDisplaySettings(nil,i,DevMode) do begin
with DevMode do begin
if (dmPelsWidth = WResolution) and
(dmPelsHeight = HResolution) and
(dmBitsPerPel = Depth) then
if ChangeDisplaySettings(DevMode,CDS_UPDATEREGISTRY) =
DISP_CHANGE_SUCCESSFUL then begin
Result := True;
Break;
end;
Inc(i);
end;
end;
end;
```

Ez a függvényünk, három bemenő paraméterrel a képfelbontás, és színmélység megváltoztatását teszi lehetővé futás közben, amennyiben a megadott értékek megfelelőnek bizonyulnak.
Mentés után, ha futtatjuk a kódot, tapasztalni fogjuk a hiányosságot is: a képfelbontás megfelelően változik (természetesen megfelelő jogosultságok, és értékek esetén), a színmélység szintén, de a képfrissítési frekvencia vissza fog esni a legalacsonyabb értékre. Példánkból azért hagytuk ki ezt, mivel a képernyők esetében nehéz eldönteni, hogy mely frissítési frekvencián képesek azok dolgozni. Amennyiben szükségesnek érezzük ezt is beállítani, akkor már érdemesebb a kódot önálló beállító ablak létrehozásával módosítani, ahol a felhasználó (a windows beállító párbeszédablakához hasonló módon) választókörökkel, lenyíló listával, vagy jelölő négyzettel állíthatja be a kívánt értékeket.
Látható azonban, hogy a példákban nem törődtünk a részletekkel, felületesen oldottunk csak meg néhány feladatot, hiszen itt a célunk csupán a lehetőségek néhány szeletének érintése volt. Maradt még egy gombnak hely, ez az utolsó gombunk is egy az előzőekhez hasonló egyszerű feladatot fog végrehajtani.

Névjegy készítése API hívással:

A Windows saját névjegy ablakának felhasználásával utolsó lépésben elkészítjük a Tipp-ek projectünk saját névjegyét:

1. Vegyük fel az utolsó gombot a formra: Name: btnNevjegy, Caption: Névjegy
2. Kattintsunk duplán a gombra, majd írjuk be a következő kódot:

```
procedure TfrmTipp.btnNevjegyClick(Sender: TObject);
begin
//A ShellAbout függvényhez is szükséges a ShellApi deklaráció!
ShellAbout(Handle,'Barhács és Társa # 2000 alá fejlesztett alkalmazásom',
        'Fejlesztő: Saját Név (2004)', Application.Icon.Handle);
end;
```

Ahogy a kommentben is látható, itt megint a ShellApi egy függvényét hívjuk meg, mely a névjegy kirajzolásáért felelős. Természetesen a névjegyünk csupán néhány elemét fogjuk tudni megváltoztatni, de már így is elég látványos elemmel bővíthetjük alkalmazásunkat. A függvény a fejrészt módosítja (a kettőskeresztig), majd a belső ablak feliratában helyezi el a további részeket. Az utolsó paraméter pedig a névjegy ablakunk ikonját fogja befolyásolni. Nézzük az eredményt:

Eredeti ablak: Általunk generált ablak:

Űrlap színének változtatása billentyűkombinációval

Ha mindent rendben találtunk, menjünk az űrlapunk Events fülére, majd válasszuk ki az OnKeyDown eseményt. A kódszerkesztőbe pedig írjuk be a következő kódot:

```
procedure TfrmTipp.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
if (shift=([ssAlt, ssCtrl])) then
        frmTipp.color:=clAqua;
end;
```

A színként definiált clAqua értékét természetesen megadhatjuk a szín kóddal is:

FrmTipp.Color:=random($FFFFFF);

Megjegyzés: a fenti példában a színkódnál véletlenértékkel dolgozhatunk, de ügyeljünk arra, hogy itt a színskála nem az RGB sorrendben dolgozik, hanem BGR (vagyis Blue, Green, Red) sorrendben.

Ha futtatnánk a programunkat, még nem történne semmi, ugyanis alapértelmezetten a Form KeyPreview tulajdonsága False, tehát ezt még meg kell változtatnunk. Ezt követően mentés, és futtatás után, ha megnyomjuk a Ctrl+Alt billentyűket, a form színe a kódunkban meghatározott színűvé változik. Természetesen további billentyűk is figyelhetőek a kódból, így lehetőségünk van akár a kombinációk több változata közül választva eltérő színeket is alkalmazni, vagy éppen valami mást (egy speciális kódrészletet futtatni) a háttérben dolgozó alkalmazásban

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. A parancssor elindításához melyik függvény használható?
   a., SendMessage
   b., ShowWindow
   c., WinExec
2. Melyik Unit része a ShellExecute függvény?
   a., SysUtils
   b., ShellApi
   c., Windows
3. Melyik Unitban található az mciSendString függvény?
   a., ShellApi
   b., MMSystem
   c., StdCtrls
4. Melyik paraméter alkalmas az ablakok láthatóságának megjelenítésére?
   a., SW_HIDE
   b., SW_MODAL
   c., SW_RESTORE

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. A GetUserName függvény visszaadja a regisztrált felhasználók listáját.
   igaz
   hamis
2. Az ExtractFilePath segítségével megtudhatjuk az alkalmazásunk elérési útvonalát.
   igaz
   hamis
3. A ShellAbout függvény segítségével testreszabható névjegyet készíthetünk.
   igaz
   hamis
4. Az ssShift, a shift billentyűzet benyomására reagál.
   igaz
   hamis
5. A WinExec függvény használatakor a windows alkalmazások teljes elérési útjának megadására szükségünk van.
   igaz
   hamis

Adatbáziskezelés I.

Alapok

Egy adatbázis-kezelő alkalmazás esetén alapvetően három különböző réteget különböztetünk meg, melyek a következők:

Az adatszolgáltatások rétege (Data Processing)

Ez a réteg felelős az adatok fizikai eléréséért, feldolgozásáért. E réteg feladata az adatbázis állományok nyitása, zárása, újadat felvitele, törlése, módosítása, indexek kezelése, zárolási konfliktushelyzetek feloldása, és így tovább.

Az alkalmazáslogika rétege (Bussines Logic)

Az alkalmazáslogika rétege az adatbázisra vonatkozó szabályok összességét tartalmazza. Gyakorlatilag ebbe a rétegbe tartoznak azok a funkciók, műveletek, amelyek meghatározzák egy adatbázis működését. Ilyen szabályok a mező illetve rekordszintű ellenőrzések (mezőszintű ellenőrzés, pl. ha egy tanuló érdemjegyeinek felvitelekor a program csak egy és öt közötti értéket enged felvinni), a hivatkozási függőségek ellenőrzése (pl. egy könyvet csak akkor lehessen eladni, ha az szerepel a könyvesbolt árukészletén) stb.

Megjelenítési réteg (User Interface)

Mely a tulajdonképpeni kezelői felület, ahol a felhasználónak lehetőséget biztosítunk az adatbázishoz való hozzáférésre.
Természetesen a fent tárgyalt rétegek a különböző adatbázis architektúrákban eltérő módokon valósulhatnak meg, attól függően, hogy a struktúra milyen lehetőségeket kínál. Nézzük tehát meg a struktúrákat:

Adatbázis architektúrák

Egygépes megvalósítás (Local Databases)

Az adatbázisoknak ez a lehető legegyszerűbb megvalósítási módja. Az alkalmazás egyetlen gépre íródott, az adatbázis és az azt feldolgozó program ugyanazon a gépen helyezkedik el, az adatbázist csak egyetlen program használja egy időben. Ebben az esetben mindhárom réteg egyazon gépen helyezkedik el.

File-kiszolgáló (File-Server) architektúra

Ebben az esetben az adatbázis állományok átkerülnek egy központi szerverre és egy időben több program is használhatja őket hálózaton keresztül. A szerver csak az adatok tárolására szolgál. Ezen megoldás esetén, ha a felhasználó akármilyen egyszerű adatműveletet akar is végrehajtani, az adatrekordoknak el kellett jutniuk a felhasználóhoz a hálózaton. Ez nagy adatforgalommal jár, ami a hálózat túlterheléséhez vezethet. Ezt a megoldást leginkább az xBase alapú (Dbase,FoxPro, Clipper..) adatbázis-kezelőkkel használják. Delphiben is írhatunk ilyen alkalmazásokat, de csak akkor érdemes, ha nincs szükség nagy teljesítményre, aránylag kevés felhasználója van a programnak, és olcsón meg akarjuk úszni a dolgot, (mivel egy adatbázisszerver nem olcsó mulatság). Szintén mindhárom fentebb tárgyalt réteg egyazon gépen helyezkedik el.

Ügyfél-kiszolgáló (Client/Server) Architektúra

Az adatbázisok implementálásának e formájában az alkalmazás két részre bomlik. Az adatok közvetlen kezeléséért egy adatbázis-szervernek nevezett software a felelős, (pl. MS SQL Server, Oracle, Informix, Sybase, InterBase stb.), míg a felhasználóval való kapcsolattartás az ügyfél program feladata. Az adatbázis-servert készen vásárolhatjuk meg, míg a kliens programot mi magunk írhatjuk meg valamilyen programozási nyelven. Az alkalmazás logikának egy részét magába az adatbázisba, a többit a kliens programba tudjuk beépíteni. Hogy ez hogyan is történik arról majd később még lesz szó. A client/server technológiában az ügyfél utasítja a szervert, pl. adatokat kér le, és erre a szerver visszaküldi az eredményt. Tehát nem kell a hálózaton a feldolgozandó adatoknak rekordról-rekordra átmenni a klienshez, hanem egy rövid parancs hatására, csak a ténylegesen kért, hasznos adatok fognak a szervertől a kliensig utazni, ezáltal jelentősen csökkentve a hálózati forgalmat. Így az adatfeldolgozást a szerver végzi a kliens parancsainak hatására. E parancsok számára kidolgoztak egy szabványos nyelvet, ez az SQL. Tehát az adatbázis-szervereket ilyen SQL parancsokkal tudjuk munkára bírni. Az adatbázis szervereknek a hálózati forgalom csökkentésén kívül számos más előnyük is van, biztosítják az egyidejű adatelérést (egyszerre nagy számú felhasználó kiszolgálására képesek), az adatbiztonságot, központilag kezeli a felhasználói jogosultságokat, stb.

Több rétegű (Multi-Tier) adatbázis architektúra

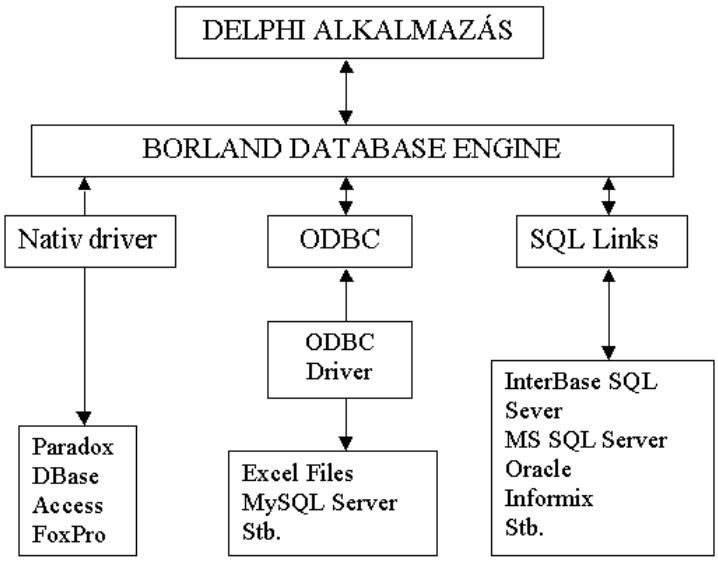
Ebben az esetben a kliens nem közvetlenül az adatbázis-szerverhez, hanem egy vagy több köztes ún. applikációs szerverhez kapcsolódik, és végül az applikációs szerver kapcsolódik az adatbázis-szerverhez. Tehát a kliensnek a középen elhelyezkedő applikációs szervertől kapják az adatokat, ezért ezt a réteget adatszolgáltatónak (Data Broker) is nevezik. Így az adatbázis logikát el lehet helyezni a középső rétegben, és a kliens feladata csak a felhasználóval való kapcsolattartás lesz. Az ilyen kliens-t "sovány" (thin) kliensnek nevezzük, hiszen a munka nagy részét az applikációs szerver végzi. Tehát ebben az esetben a három réteg, fizikailag is három különböző helyen helyezkedhet el.

Delphiben természetesen lehetőségünk van a fentebb említett bármelyik architektúrát felhasználva adatbázisos alkalmazást készíteni. Nézzük meg, hogyan lép kapcsolatba a Delphi a különböző adatbázisokkal.

A Borland Database Engine (Adatbázis motor)

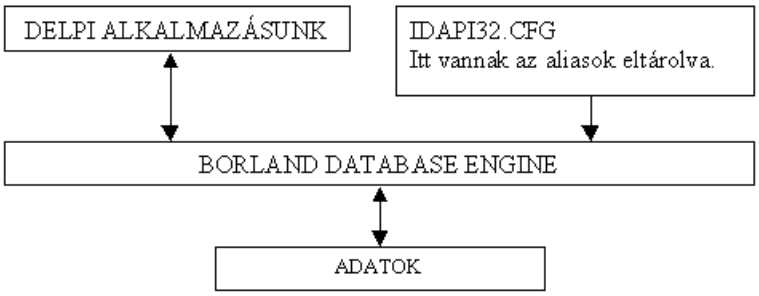
Ahhoz, hogy egy Delphiben irt alkalmazás meg tudjon nyitni egy helyi adatbázist, illetve, hogy csatlakozni tudjon egy adatbázisszerverhez, szüksége van a Borland Database Engine-re (továbbiakban BDE). Ezért ha egy adatbázisos alkalmazást készítünk, és azt telepítjük egy felhasználó gépére, ahhoz hogy működjön a program, a BDE-t is telepíteni kell ugyanarra a gépre. A BDE egységes felületet biztosít a különböző formátumú adatbázisok eléréséhez, nem kell ismernünk az egyes rendszerek sajátosságait. A BDE gyakorlatilag DLL-ek és utility-k gyűjteménye, melyek segítségével különböző adatbázisokat érhetünk el egységesen Borland fejlesztőeszközökkel készített programunkból. (pl. a C++ Builder, vagy az IntraBuilder is a BDE-t használja) A Delphi minden verziója tartalmaz ún. STANDARD drivereket, melyekkel a lokális adatbázisokat (Paradox, dBase) érhetjük el. Ahhoz hogy különböző adatbázisszerverekhez (Oracle, Sybase, InterBase stb.) is csatlakozni tudjunk a client/server verzióra van szükség. Az adatbázis-szerverek

eléréséhez a BDE az ún. SQL-links csomagot használja. Az SQL-links gyakorlatilag kiegészítő drivereket tartalmaz a BDE-hez.



BDE Aliasok (Alnevek)

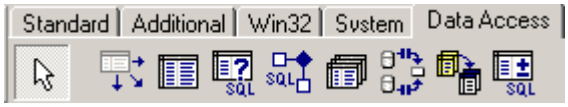
A BDE álneveket (alias) használ a különböző adatbázisokra való hivatkozáskor. Az alias gyakorlatilag paraméterek halmaza, ami egyszerűbb esetben, lokális adatbázisoknál az adatbázis elhelyezkedését és típusát tartalmazza, adatbázisszerverek esetén pedig egy csomó plusz paraméter megadható, pl. a megnyitás módja, a szervernév, felhasználónév stb. Amikor elkészítjük az alkalmazásunkat, akkor abban alias-sal hivatkozhatunk a használt adatbázisra. Így ha később pl. megváltozik az adatok elérési útvonala, az nincs fixen belefordítva a programunkba, hanem egyszerűen megváltoztathatjuk azt a későbbiek során bármikor. Alias-t a BDE Administrator-ral, a Database Explorer-rel, (A BDE Administrator az adatbázismotor konfigurációs programja, mellyel aliasokat hozhatunk létre, módosíthatunk vagy törölhetünk. A Database Explorer egy segédprogram mellyel aliasokat kezelhetünk, adatbázisokat nézhetünk meg, módosíthatunk, sql lekérdezéseket futtathatunk stb.) de akár saját magunk programból is létrehozhatunk. A létrehozott alias a BDE saját konfigurációs állományában (IDAPI32.CFG) kerül elmentésre, és mindaddig megmarad míg nem töröljük. Miután elindítottuk a BDE Administratort, már alapesetben is látható lesz néhány alias, melyek a Delphi példaadatbázisaira mutatnak.



Adatbáziskezelési komponensek:

Maguk a komponensek két fő csoportra bonthatóak: látható, és nem látható komponensekre. A nem látható komponensek fogják a kapcsolatokat a háttérben felépíteni, majd az összeköttetést biztosítani a látható komponenseink, és a háttérben lévő adatbázis között. Nézzük meg ezeket az elemeket:

Data Access komponenspaletta: (Adatelérési komponensek)



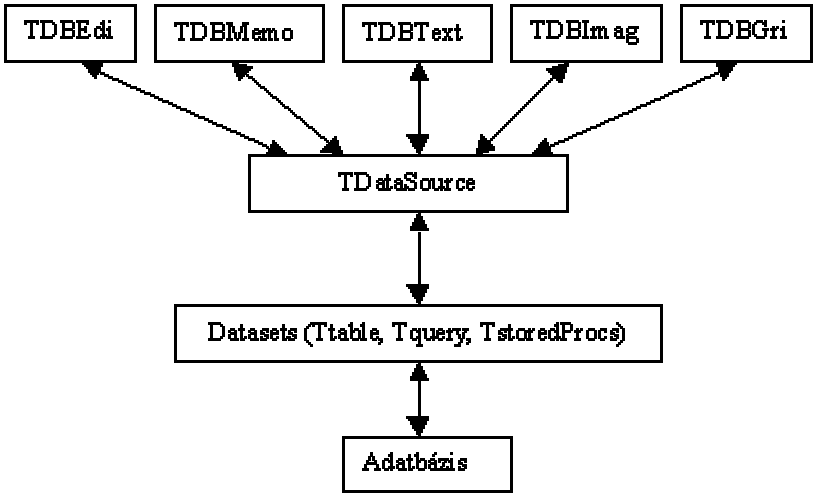
A nem látható elemek, melyek a háttérben a kommunikációt fogják biztosítani a kezelői felületen elhelyezett vizuális elemek, és az adatbázisunk között.
DataSource: adatforrás beállítása.
Table: adatbázisunk egy konkrét táblája.
Query: lekérdezés létrehozása.
StoredProc: tárolt eljárások használata.
DataBase: adatbázis elérése.
Session: adatbázis menedzselése.
BatchMove: Csoportos parancsműveletek kiadása.
UpdateSQL: Az adatok megváltozása utáni azonnali frissítéshez.

Az elemek elhelyezésével építhetőek ki a kívánt elérések a háttérben futó adatbázisainkkal. Mivel ezek a komponensek nem láthatóak, így csak arra adnak lehetőséget, hogy alapjai legyenek a vizuális elemek adateléréseinek.

Data Controlls komponenspaletta: (Adatmegjelenítési komponensek)

A tényleges megjelenítésért felelős elemek, melyek a felhasználók számára is láthatóak:
DBGrid: Csoportos megjelenítő.
DBNavigator: a rekordléptetések, vezérlések eleme.
DBText: Címkék az adatbázishoz.
DBEdit: beviteli mezők az adatbázisból.
DBMemo: feljegyzések az adatbázisból.
DBImage: Képeink megjelenítése az adatbázisból.
DBListBox: Lista elemekhez.
DBComboBox: Lenyíló listákhoz.
DBCheckBox: jelölőnégyzetek
DBRadioGroup: Csoportos elemek (választókör) megjelenítése
DBLookupListBox: Listaelem, sorforrással, és adattárolóval.
DBLookupComboBox: Lenyíló lista saját sorforrással, és tárolóval (mint az előző).
DBRichEdit: Egyszerű szövegszerkesztő a Memo típusú adatokhoz.
DBCtrlGrid: Önálló vezérlővel rendelkező felület a rekordok áttekintéséhez.
DBChart: Diagram megjelenítésének lehetősége.

Az adatelérési és adatmegjelenítési komponensek nem tudnak közvetlenül egymáshoz kapcsolódni. Ahhoz hogy az adatelérési komponens által reprezentált adatokat meg tudjuk jeleníteni egy adatmegjelenítési komponenssel, szükség van az ún. TDataSource (adatforrás) komponensre. Először a TDataSource kapcsolódik egy TDataSet komponenshez, majd ezután egy adatmegjelenítési komponens DataSource tulajdonságát tudjuk ráállítani az adatforrásunkra.
Így jelenik meg egy adat a felhasználónak adatmegjelenítési komponens segítségével:



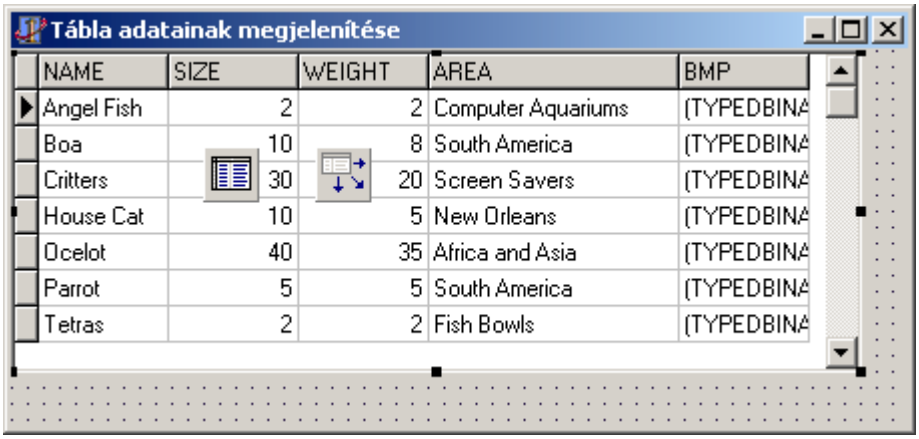
Miután nagy vonalakban megismerkedtünk az adatbázis-kezelés Delphi-s alapjaival, ideje, hogy megnézzük mindezt a gyakorlatban is. Készítsük el első adatbázis munkánkat. Ehhez persze először létre kellene hoznunk egy adatbázist, vagy legalábbis egy táblát, melynek adatait megjeleníthetjük. Hogy ne kelljen ezzel töltenünk az időt, a Borland elhelyezett néhány próbaadatbázist a Delphi mellé, melyeket elérhetünk, és használhatunk is. Első feladatunk tehát annak megértése lesz, hogy miképpen jelennek meg az adatok a felhasználó számára:

Feladat: Adatbázis készítés: (ANIMALS)

Ebben a feladatban nézzük meg, milyen kapcsolatokon keresztül (az komponensek milyen beállításaival) jelennek meg egy létező adatbázis tábla adatai a felhasználó számára. Hozzuk létre a Delphi könyvtárunkban az ANIMALS alkönyvtárat.

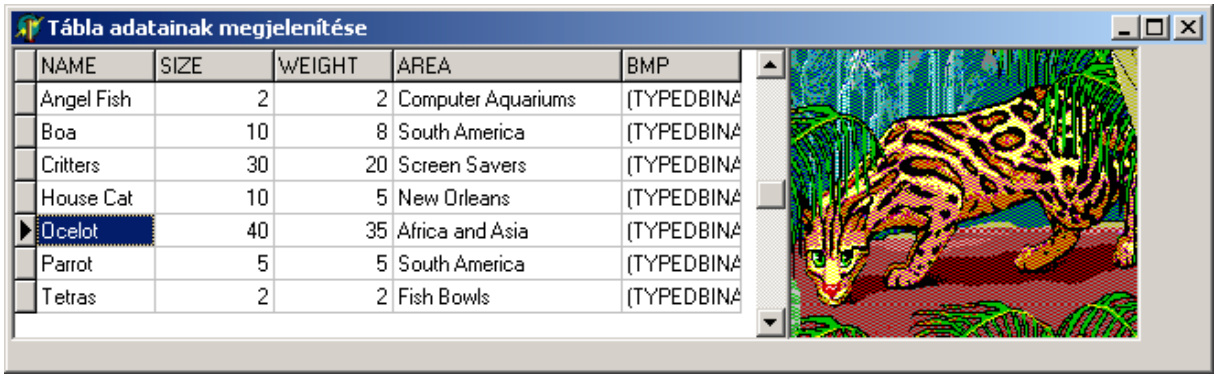
1. Indítsuk el a Delphit, a form Name: frmAnimals, Caption: Tábla adatainak megjelenítése.
2. Save All: Uanimals, Panimals.
3. A DataAccess panelről helyezzünk el egy Table komponenst a formunkra (nem látható, tehát bárhová kerülhet). A Table DatabaseName tulajdonságát módosítsuk: legyen DBDEMOS. Amit most kiválasztottunk tulajdonképpen nem más, mint a Delphi Alias nevek közül az egyik gyári beállítás. Ha az elérési út (alias) beállításra került, akkor hozzá tudjuk rendelni táblakomponensünkhöz a megjeleníteni kívánt táblát: TableName: animals.dbf Ezzel az elérési úton található dbf (Data Base File) állományok közül kiválasztottunk egyet. Ahhoz, hogy működjön majd a kapcsolatunk, a táblánk Active tulajdonságát True-ra kell még állítanunk. Ezzel élővé tettük a kapcsolatot.

4. Vegyünk most fel a táblánk mellé (szintén a DataAccess palettáról) egy DataSource elemet, és állítsuk át a DataSet tulajdonságát az egyetlen lehetséges Table1-re. Természetesen hiába mentenénk és futtatnánk, még nem történne semmi, hiszen az elemek nem látható elemek.
5. Lépjünk át a DataControlls palettára, és vegyünk fel egy megjelenítésért felelős elemet: a DBGrid-et. Tulajdonképpen csupán egyetlen tulajdonságot kell beállítanunk: a DataSource legyen az egyetlen lehetséges elemünk: a DataSource1. Már fejlesztés közben meg kell hogy jelenjenek a rekordok a Gridben (ezért állítottuk True-ra a tábla Active tulajdonságát). Ha itt nem látom az adatokat, akkor biztosan nem fog megjelenni futásidőben sem, tehát most kell megkeresnem az esetleges hibákat.



Ha mentettünk, és futtatunk, látni fogjuk, hogy a BMP mezőnk (mely valószínűleg egy képet fog tartalmazni) a Grid-ben nem jelenik meg. Természetesen futásidőben már a többi adatunk szerkeszthető (lehetőség szerint ne írjuk felül a példaadatbázist!). A Grid előnye: alkalmazkodni fog a táblákban szereplő mezőkhöz, vagyis annyi oszlop jelenik majd meg, amennyi a táblánkban van. Nagyon egyszerűen jeleníthetünk meg a segítségével adatrekordokat, eltérő táblák adatait. De hogy jelenhet meg a kép?

6. Helyezzünk el a DBGrid mellé egy DBImage komponenst, majd a DataSource legyen DataSource1, és a DataField legyen a BMP. Megint meg kellene jelennie már fejlesztés alatt a képnek, és futásidőben pedig a Gridben léptetve az adatainkat a kép annak megfelelően változik majd.



Ha mindent megfelelőnek találtunk mentsük a munkánkat.
Nézzük ezek után, milyen fontosabb közös tulajdonságai vannak az adatelérési komponenseknek:

Adatelérési komponensek fontosabb közös tulajdonságai (property-k):

Active	Egy adathalmaz megnyitására, zárására szolgál. Ha True, akkor van nyitva az adathalmaz, egyébként zárva.
Bof	Értéke True, ha a kurzor az adathalmaz elején van, egyébként False.
CachedUpdates	Ha True, akkor az adatbázison történő módosítások egy cache-ben maradnak a kliens gépen, és egy tranzakció keretében egyszerre lehet a módosításokat érvényesíteni. Ha False, akkor minden módosítás azonnal az adatbázisban is megtörténik.
CanModify	Azt jelzi, hogy módosítható-e az adathalmaz adata.
DataSource	Annak a DataSource komponensnek a neve, amely az adathalmazhoz kapcsolódik (már ha van ilyen).
DatabaseName	Az adatbázis neve, melyhez kapcsolódik a komponens. (lehet fizikai út, vagy alias)
Eof	Értéke True, ha a kurzor az adathalmaz végén van, egyébként False.
FieldCount	Az adathalmaz mezőinek számát adja.
Fields	Egy tömb, mely Tfield objektumokat tartalmaz.
Filter	Szűrési feltétel adható meg, az adathalmaz rekordjaira.
Filtered	Ha True, akkor csak a szűrési feltételnek megfelelő rekordok jelennek meg, a False akkor az összes rekord
FilterOptions	A szűrés tulajdonságainak beállítása.
Found	Jelzi, ha egy keresés sikeres volt.
Modified	Jelzi, ha egy rekord módosítása megtörtént.
Name	A komponens neve.
RecNo	Az aktuális rekord számát adja vissza az adathalmazban.
RecordCount	Az adathalmaz összes rekordjának számát adja.
State	Az adathalmaz aktuális állapotát jelzi (pl. dsEdit, dsInsert stb)

Természetesen ezeken kívül egyes komponensek további egyéb tulajdonságokkal rendelkeznek, de ezek egyenkénti részletes felsorolása megint nem képezi az elsajátítandó ismeretek részét. Egy adott tulajdonság megváltoztatása természetesen attól függ, hogy éppen mit szeretnénk az adatbázis adataival tenni, így objektumonként eltérő tulajdonság-beállításokra is szükségünk lehet. Segítséget ismét a súgó adhat, itt elég, ha a tulajdonság (property) mezőjén megnyomjuk az F1-et. Nézzük ezek után a legfontosabb közös metódusokat, és eseményeket:

Adatelérési komponensek legfontosabb közös metódusai:

Append	Létrehoz egy üres rekordot és az adatbázis végéhez fűzi.
AppendRecord	Létrehoz egy üres rekordot, és azt feltölti az átadott értékekkel.
Cancel	Egy rekord editálásánál lehet érvényteleníteni a módosításokat.
ClearFields	Törli az adott rekord összes mezőjének értékét.
Close	Az adathalmaz lezárása.
Delete	Törli az aktuális rekordot.
DisableControls	Letiltja az adatelérési és az adat-megjelenítés komponens közötti kapcsolatot.
Edit	Engedélyezi az aktuális rekord editálását.
EnableControls	Engedélyezi az adatelérési és adat-megjelenítési komponens közötti kapcsolatot.
FieldByName	A megadott mezőnév Tfield komponensére hivatkozhatunk a segítségével.
FindFirst	Megkeresi a szűrési feltételnek megfelelő első rekordot.
FindNext	Megkeresi a szűrési feltételnek megfelelő következő rekordot.
FindLast	Megkeresi a szűrési feltételnek megfelelő utolsó rekordot.
FindPrior	Megkeresi azt a megelőző rekordot, amely megfelel a szűrési feltételnek.
First	A kurzort az adathalmaz első rekordjára állítja.
FreeBookmark	Könyvjelző létrehozásakor lefoglalt memória felszabadítására szolgál.
GetBookmark	Az aktuális rekordon elhelyez egy könyvjelzőt.
GetFieldNames	Visszaadja egy adathalmaz mezőinek nevét egy string-listában.
GotoBookmark	A megadott könyvjelző által jelölt rekordra ugrik.
Insert	Beszúr egy rekordot az adathalmazba, az aktuális pozícióban.
InsertRecord	Beszúr egy rekordot és feltölti az átadott értékekkel.
Last	Az adathalmaz utolsó rekordjára ugrik.
Locate	Keresés. Ha van találat, a feltételnek eleget tevő rekord lesz az aktuális.
Lookup	Keresés, amely a keresett értéket adja vissza, a rekordmutató nem mozdul el.
MoveBy	A rekordmutatót az aktuális pozícióból megadott értékkel elmozgatja.
Next	A rekordmutatót a következő rekordra viszi.
Open	Megnyit egy adathalmazt.
Post	Editálás után ezzel a metódussal lehet az adatok módosítását véglegesíteni.
Prior	A rekordmutatót a megelőző rekordra viszi.
Refresh	Az adathalmaz által reprezentált adatokat frissíti az adatbázisból.
SetFields	Egy adathalmaz egy rekordjának összes mezőjét egyszerre lehet módosítani a metódus segítségével.

Fontosabb közös események (Events):

AfterCancel	Rekord editálásakor Cancel után hívódik meg.
AfterClose	Adathalmaz zárása után generálódik.
AfterDelete	Rekord törlése után generálódik.
AfterEdit	Rekord editálása után generálódik.
AfterInsert	Rekord beszúrása után generálódik.
AfterOpen	Adathalmaz megnyitása után generálódik.
AfterPost	Egy rekord módosításának Post-olása után generálódik.
BeforeCancel	Rekord editálásakor Cancel előtt hívódik meg.
BeforeClose	Adathalmaz lezárása előtt generálódik.
BeforeDelete	Rekord törlése előtt generálódik.
BeforeEdit	Azelőtt generálódik, mielőtt egy adathalmaz editálható állapotba kerülne.
BeforeInsert	Rekord beszúrása előtt generálódik.
BeforeOpen	Azelőtt generálódik, mielőtt egy adathalmaz megnyitott állapotba kerül.
BeforePost	A Post metódus meghívása előtt generálódik.

OnCalcFields	Egy mező számításakor generálódik. (ide kell elhelyezni a képletet, ami a számított mező értékét számolja)
OnDeleteError	Egy rekord törlésének hibájakor generálódik.
OnEditError	Akkor generálódik, ha editálásnál valami hiba lép fel.
OnFilterRecord	Adathalmazok szűrésére használjuk.
OnNewRecord	Egy új rekord felvitelekor generálódik.
OnPostError	Akkor generálódik, ha a Post metódus meghívásakor hiba lép fel.
OnUpdateError	Akkor generálódik, ha akkor lép fel hiba, miközben a Cached Updates-el adatokat töltünk fel az adatbázisba.
OnUpdateRecord	Akkor generálódik, amikor a Cached Updates egy rekordot módosít.

Field Editor

Egy adatelérési komponensen kettőt kattintva, vagy jobb gombot nyomva és a Field Editor menüt választva jelenik meg a mezőszerkesztő. Ez alapesetben üres. Újabb jobb gomb hatására bejön egy menü, ahol mezőket adhatunk az adathalmazhoz, törölhetünk belőle, származtatott mezőket hozhatunk létre, és különféle editálásra nyílik lehetőségünk. A mezőszerkesztőben lévő mezők közül tetszőlegeset kiválasztva, az Object Inspektorban megjelennek az adott mezőhöz tartozó Tfield objektum tulajdonságai, eseményei, amiket ezután tetszés szerint beállíthatunk a megfelelő értékekre.

Származtatott mezők

Kétféle származtatott mezőt hozhatunk létre.

Lookup Fields (Kikeresett mezők) Olyan mező, amelynek értékét egy másik táblából keressük ki. Pl. adott egy árucikkeket tartalmazó tábla, amelyben egy kétbetűs azonosító jelzi az adott árucikkhez tartózó szállítót, és van egy másik tábla, amely csak a szállítókat tartalmazza. Ekkor az azonosító alapján a program mindig kikeresi a megfelelő árucikkhez az adott szállító szükséges adatait. (pl név, cím stb.)

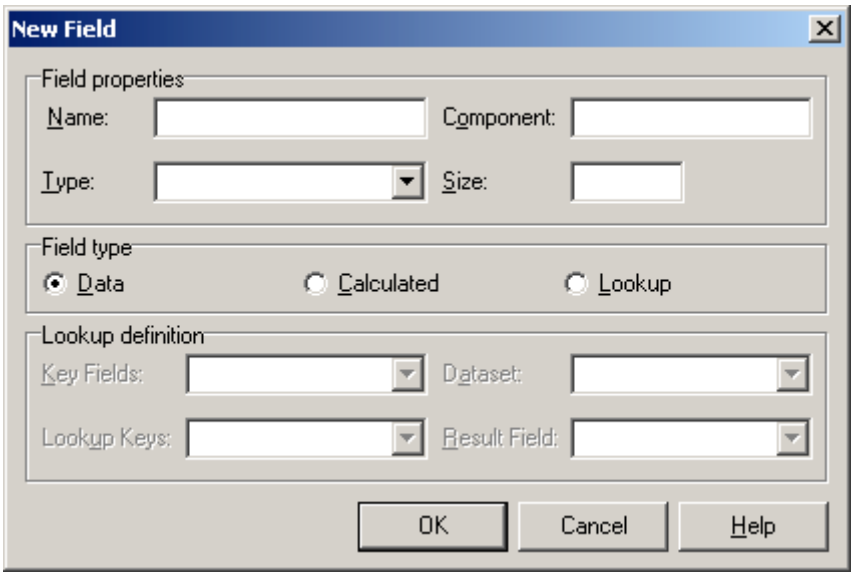
Calculated Fields (Számított mező) Értéke nem szerepel a táblában, a program meglévő mezők értékeiből fogja kiszámolni. Pl. ha megvan a nettó ár és az áfa, akkor fölösleges a bruttó árat tárolni, csak fölöslegesen növelnénk a tábla méretét redundanciát okozva, ahogy ezt korábban már az adatbázis-kezelés alapjainál megtanultuk.

Lookup Fields (Kikeresett mezők) létrehozása

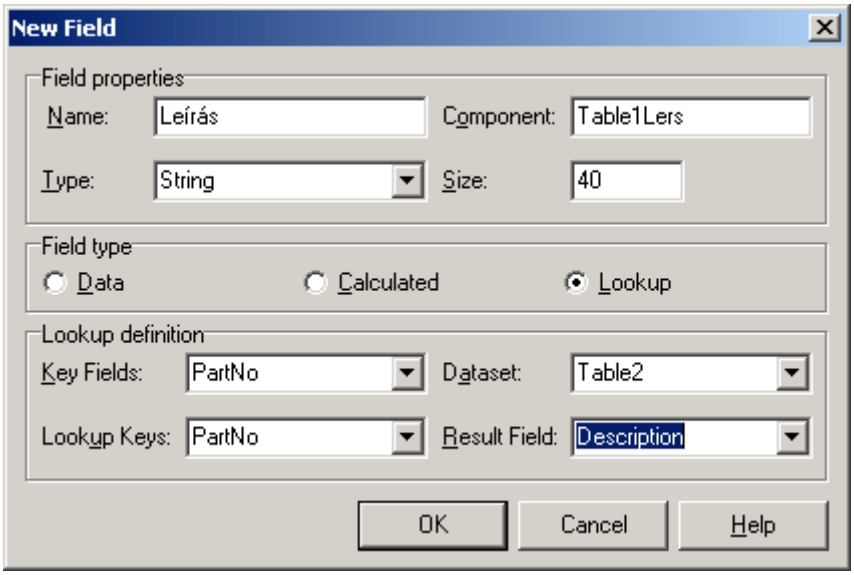
Kikeresett mezőt a Field Editorral hozhatunk létre. Nézzük meg a gyakorlatban hogyan:

Hozzunk létre a Delphin belül egy DBKERES könyvtárat, majd indítsuk a Delphit.

1. A form neve: frmDemos, Caption: Példák számított, és keresett mezőkre. Mentés: Ukeres, Pkeres.
2. Két táblára lesz szükségünk, az egyik az alaptábla, amelyben létrehozzuk az új mezőt, a másik, amelyikből származtatjuk a létrehozott mező értékeit. Ehhez a Delphi-hez adott példa táblákat használjuk fel. Tegyünk két Table, komponenset a Form-ra. A DatabaseName mindkettőnél legyen a DBDEMOS, az első táblát irányítsuk az items.db-re, a másikat a parts.db-re.
3. Az items tábla mezőit jelenítsük meg egy rácsban. Ehhez vegyünk fel egy DataSource komponenst, DataSet property-ét állítsuk az items.db-re irányított első táblára.
4. Vegyük fel a DBGrid-et is, DataSource property-ét az adatforrás komponensre (DataSource1).
5. Mindkét táblát állítsuk aktívra, ekkor a rácsban meg kell jelennie az items táblának.
6. Az items táblában létrehozunk egy számított mezőt, ami a parts táblából a leírást (description mező) tartalmazza. Ehhez kattintsunk duplán az items táblára irányított table komponensre, hogy bejöjjön a mezőszerkesztő. Itt jobb gombot nyomva, bejön egy menü, ahol válasszuk, az Add Fields menüt, és adjuk az összes mezőt az adathalmazhoz. Ha ez megvan, újabb jobb gomb után válasszuk a New Field menüt. A képernyőn a következőt kellene látnunk:



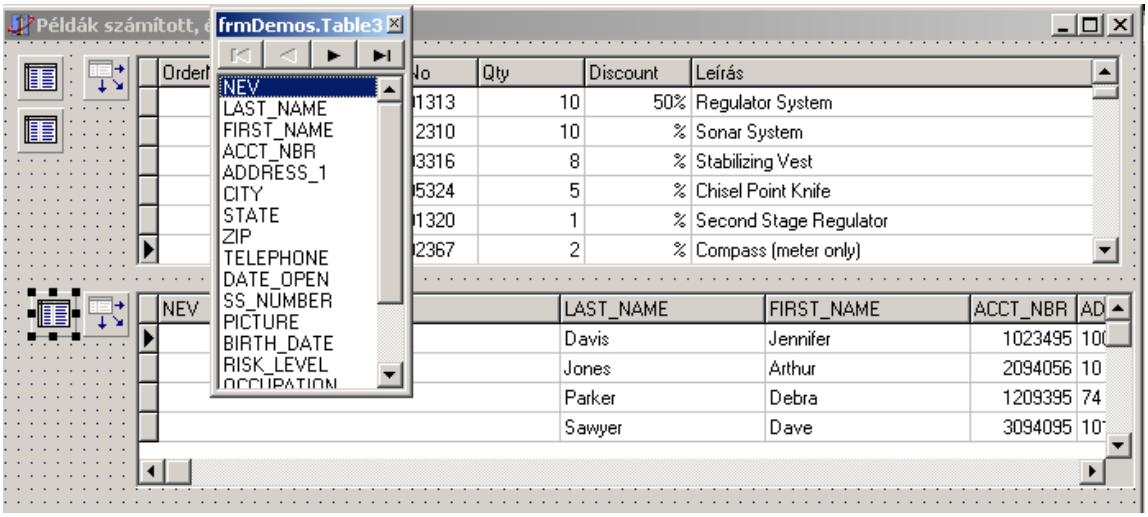
Ekkor a következő dolgokat állítsuk be:



Ha ezzel megvagyunk, OK gombot nyomva, máris meg kell hogy jelenjen a rácsban egy új, Leírás nevű mező. Mentsünk, majd futtassunk.

Calculated Fields (Számitott mezők létrehozása)

Számított mező létrehozásához a mezőszerkesztőben a New Field menüpontban a Field Type Calculated legyen. Ezen kívül még a mező nevét és típusát kell megadnunk. Az új mező kiszámításának képletét a tábla OnCalcFields eseményjellemzőjébe kell beírni.
Folytassuk az előző projectet, tegyünk a form-ra egy újabb Table, DataSource, és egy DBGrid komponenst. A DatabaseName legyen a DBDEMOS, a TableName most a clients.dbf. Az előbbihez hasonló módon jelenítsük meg a rácsban a table tartalmát. A mezőszerkesztőben a Field Type legyen Calculated, a neve pedig NEV.
Ezt követően (mivel a listánk legvégére került) érdemes még a Drog & Drop módszerrel a mezőszerkesztőben az első helyre helyezni azt. Ha megfelelően jártunk el, ezt kellene látnunk:



Bezárhatjuk a mezőszerkesztőt, majd keressük meg a Table3 komponensünk tulajdonságlapján az Events eseményt.
A Table OnCalcField eseményébe írjuk be a következő kódot:

```
procedure TfrmDemos.Table3CalcFields(DataSet: TDataSet);
begin
Dataset.FieldByName('NEV').AsString:=Dataset.FieldByNAme('FIRST_NAME').AsString+
        ' '+Dataset.FieldByNAme('LAST_NAME').AsString;
end;
```

Ha mindent jól csináltunk, akkor miközben futtatjuk a programot a rácsban meg kell jelennie egy NEV mezőnek, ami a teljes nevet állítja elő egy számított mezőben a keresztnév és a vezetéknév alapján. Kérjünk mentést, majd bezárhatjuk az alkalmazásunkat.

Rekordokon végzett műveletek

Rekord szerkesztése
Egy rekord nem minden esetben szerkeszthető. Ha pl. egy táblát csak olvashatóként nyitottunk meg, vagy egy másik felhasználó zárolta előlünk az adott rekordot, akkor azt nem tudjuk szerkeszteni. Erről a CanModify property értékének megvizsgálásával győződhetünk meg. Ha a CanModify property True, akkor szerkeszthető számunkra a rekord. Ahhoz hogy egy rekordot szerkeszthessünk, az adathalmazt dsEdit állapotba kell állítanunk az Edit metódus segítségével. Módosításaink elmentésére a Post metódus szolgál. Ha nem hívjuk meg a Post metódust, de átlépünk egy másik rekordra akkor is elmentődnek a módosítások. Ha valamilyen okból mégse akarjuk megtartani az új értékeket, akkor a Cancel metódus hatására megszakad az aktuális editálás és a rekord mezői, megtartják régi értéküket. A SetField metódus segítségével az aktuális rekord összes mezője egyszerre módosítható.
Rekord Törlése
Egy rekord törlésére a Delete metódus szolgál. Pl. a Table1.Delete hatására törlődik az adatbázis tábla aktuális rekordja.
Új rekord felvitele
Új rekord felvitelére az append, appendrecord, insert, insertrecord metódusok szolgálnak. Az append egy új üres rekordot fűz az adathalmaz végéhez, az insert pedig a rekordmutató aktuális pozíciójába szúr be egy üres rekordot. Az

appendrecord-ot és az insertrecordot használva megadhatók az új rekord mezőinek értékei is.

Új rekord felvitele Apend-det használva

```
With Table1 Do
Begin
 Append;
 FieldByName('NEV').AsString:= Edit1.Text;
 FieldByName('CÍM').AsString:= Edit2.Text;
 Post;
End;
```

Új rekord felvitele InsertRecordot használva

```
Table1.InsertRecord('Barhács','és Társa',null, null, null);
```

Ha egy rekord egy mezőjét nem akarjuk feltölteni értékkel, akkor az érték helyett írjunk null-t az adott mező helyére.

Keresés az adathalmazban

A következő metódusok szolgálnak keresésre:

Locate

Ha van találat, akkor logikai igaz értéket ad vissza a metódus, és az első a keresési feltételnek eleget tevő rekordra áll a rekordmutató. Indexelt és index nélküli mezőkre is kereshetünk a segítségével. Két keresési opció beállítása lehetséges, az egyik hogy megkülönböztesse-e a kis és nagybetűket (loCaseInSensitive), illetve, hogy csak teljes egyezés esetén jelezzen találatot, vagy részleges egyezés esetén is (loPartialKey). Arra is lehetőség van, hogy egyszerre több mezőre adjunk keresési feltételt.

Példák a Locate használatára:

```
With Table1 Do
 Begin
 If  Locate('LAST_NAME','Gábor',[loCaseInSensitive]) Then
  {a LAST_NAME mezőben keressük a Gábor nevet,
   nem különböztetjük meg a kis és nagybetűket}
     Begin
     {Találat esetén itt dolgozhatjuk fel a rekordot.}
     End
  Else ShowMessage('Nincs találat!')
```

Keresés két feltétellel:

```
With Table1 Do
 Begin
 If  Locate('LAST_NAME;FIRST_NAME',
VarArrayOf(['Viktória','Kis']),[loCaseInSensitive]) Then
{Mivel két értéket kell átadnunk, a VarArrayOf függvénnyel létrehozunk egy
Variant tömböt.
  Begin
  {Találat esetén itt dolgozhatjuk fel a rekordot}
  End
 Else ShowMessage('Nincs találat!');
```

LookUp

A LookUp nem mozgatja el a rekordmutatót, hanem a keresett mező értékét adja vissza, egyébként úgy működik, mint a Locate.

Lakhely:=Table1.LookUp('LAST_NAME','Anita','City');

FindKey

Ez a metódus csak a Table komponensnél használható. Az aktuális index oszlopában keres a megadott feltétel szerint. A Locate-hez hasonlóan találat esetén a rekordmutatót az adott rekordra mozgatja és logikai igaz értékkel tér vissza.

```
With Table1 Do
 Begin
 If FindKey(['Pécs']) Then
  Begin
  {Találat esetén itt dolgozhatjuk fel a rekordot}
  End
  Else ShowMessage('Nincs találat!')
```

FindNearest

Ez a metódus is csak a Table komponensnél használható. Hozzávetőleges keresést végez az aktuálisan indexelt oszlopban. Mindig van találat, hiszen ha nincs is meg a pontos kifejezés, az ahhoz legközelebbi rekordra áll.

```
With Table1 Do
Begin
If FindNearest(['Pécs']) Then
  Begin
  {Találat esetén itt dolgozhatjuk fel a rekordot}
  End
  Else ShowMessage('Nincs találat!')
```

Adathalamazok szűrése

Ha egy adathalmaz tartalmát valamilyen szempont szerint le kell szűkítenünk, pl. csak a Pécs lakhelyűeket akarjuk megjeleníteni a barátainkat tartalmazó adathalmazból, akkor szűrést kell alkalmaznunk. Erre Delphiben két féle lehetőségünk van.

1. a Filter property segítségével, vagy
2. Az OnFilterRecord eseményjellemző felhasználásával.

Ha a Filter property-t használjuk, akkor meg kell adni azt a kifejezést, ami alapján szűrjük az adathalmazt, és a Filtered jellemzőt true-ra kell állítani.

A következő operátorokat használhatjuk:

Operátor	*Jelentése*
<	Kisebb mint
>	Nagyobb mint
=	Egyenlő
<>	Nem egyenlő
>=	Nagyobb vagy egyenlő
<=	Kisebb vagy egyenlő
AND, OR, NOT	Logikai operátorok

Egy pár példa filter kifejezésre:

```
VAROS = 'Pécs'
VAROS = 'Pécs' and DATUM < '1/1/99'
VAROS = 'Pécs' and DATUM > '1/1/99'
VAROS = 'P*'
```

Ahhoz, hogy a szűrési feltételünk életbe lépjen, a Filtered property-t ne felejtsük True-ra állítani.

Lehetőség van arra, hogy bizonyos beállításokkal befolyásoljuk a szűrés folyamatát. Erre az FilterOption property kell felhasználnunk. A foCaseInSensitive hatására a szűrés folyamán nem lesznek megkülönböztetve a kis és nagybetűk. Ha a FilterOptions halmaznak eleme a foNoPartialCompare is, akkor ha a feltételben *

karaktert használunk, az joker karakterként fog funkcionálni, egyébként sima * karakternek számit.
A másik lehetőség, hogy adathalmazunkat szűrjük, az OnFilterRecord eseményjellemző használata.
Az OnFilterRecord eseményjellemzőt használva lehetőség nyílik arra, hogy egy rekordon belül különböző mezőértékeket hasonlítunk össze. Az eseményjellemzőt használva vigyázni kell arra, hogy ne végezzünk olyan műveletet, ami az esemény ismételt bekövetkezéséhez vezetne.
Példa az OnFilterRecord használatára:

```
procedure TForm1.Table1FilterRecord(DataSet: TDataSet;
var Accept: Boolean)
var Ertek : Integer;
begin
Ertek := Table1.FieldByName(`CustNo').Value;
Accept := (Ertek = 1384);
end;
```

Ahhoz, hogy ez a módszer működjön, szintén igazra kell állítani a Filtered property-t.
Ha egy szűrt adathalmazban akarunk keresni, akkor használhatóak a következő metódusok:
FindFirst - az első szűrési feltételnek megfelelő rekordot keresi
FindNext - a soron következő szűrési feltételnek megfelelő rekordot keresi
FindPrior - a megelőző szűrési feltételnek megfelelő rekordot keresi
FindLast - az utolsó szűrési feltételnek megfelelő rekordot keresi

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Hány részre bonthatjuk a Delphiben az adatbázis architektúrákat?
   a., 2
   b., 3
   c., 4
2. Melyik elem nem része az adatelérési komponenseknek?
   a., StoredProc
   b., BatchMove
   c., DBGrid
3. Melyik tulajdonsággal nem rendelkeznek az adatelérési komponensek?
   a., Active
   b., Eof
   c., Open
4. Mire alkalmas a Delete metódus?
   a., Tábla törlésére
   b., Mező törlésére
   c., Rekord törlésére

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. A Lookup Fields értékét egy másik tábla adatai alkotják.
   igaz
   hamis
2. Egy rekordot csak abban az esetben szerkeszthetünk, ha a CanModify értéke igaz.
   igaz
   hamis
3. Adathalmazok szűrésére a Delphiben 5 lehetőségünk is van.
   igaz
   hamis
4. A ClearFields metódus törli az aktuális rekord valamennyi mezőjét.
   igaz
   hamis
5. A DBNavigator az adatbázisunkban történő léptetések vezérlőeleme.
   igaz
   hamis

Adatbáziskezelés II.

Mezőobjektumok

A mezőobjektumok a Tfield osztály leszármazottai. Azt, hogy egy mezőhöz milyen mezőobjektum tartozik, a mező típusa határozza meg. Pl. karakteres mezőhöz TstringField, numerikushoz TintegerField stb. tartozhat. Egy mezőre többféleképpen lehet hivatkozni, a hozzá tartozó mezőobjektum útján, vagy a mezőnév alapján. Nézzük ezeket:

FieldValues property vagy a FieldByName metódus segítségével:

Pl.:
```
Table1.FieldValues['VAROS']:='Budapest';
Table1.FieldByName('VAROS').Value:='Szeged';
```

A Fields tömb segítségével:

Pl.: `Table1.Fields[0].Value:=152;`

A mezőobjektum neve alapján:

(a mezőobjektum neve az adathalmaz és a mező nevéből képződik)
Pl.: `Table1NAME.Value:='János';`

Mezőobjektumok fontosabb tulajdonságai:

Alignment	Mező tartalmának igazítása megjelenítési komponensen belül.
ConstraintErrorMessage	Ha a CustomConstraint jellemzőbe írt feltétel nem teljesül, az ide írt szöveg jelenik meg. Pl. "Csak 'I' vagy 'N' irható be!"
CustomConstraint	Csak az itt megadott feltételnek megfelelő értékek írhatók be a mezőbe (mezőszintű ellenőrzés).
DefaulExpression	Alapértelmezett érték adható meg egy új mező felvitelekor. Tehát ha egy új rekordot viszünk fel az adathalmazba, és nem adunk neki értéket a mezőnek, akkor az itt beállított értéket veszi fel a mező.
DisplayFormat	Megjelenítéskor formátum adható meg. Pl.: "0 db" esetén a nulla helyére behelyettesíti az aktuális mezőértéket, és mindig utána írja a "db" szót.
DisplayLabel	A mező címkéje. Ez jelenik meg például DBGrid-ben a fejlécként.
DisplayWidth	A mező szélessége.
EditMask	Bemeneti-maszk editáláskor.
EditFormat	Editáláskor formátum adható meg. Pl.: "0 db" esetén a nulla helyére behelyettesíti az aktuális mezőértéket, és mindig utána írja a "db" szót.
FieldName	A mező táblabeli neve.
IsNull	Le lehet kérdezni, hogy egy mező üres-e
MinValue	Számok esetén a mező megengedett értékének alsó határa adható meg vele.
MaxValue	Számok esetén a mező megengedett értékének felső határa adható meg vele.
Name	A mezőobjektum neve.
ReadOnly	Ha True, akkor csak olvasható, nem szerkeszthető a mező
Requied	Ha True, akkor új rekord felvétele esetén az adott mezőt nem lehet üresen hagyni, mindig értéket kell kapnia.
Value	Egy mező értékét olvashatjuk ki, vagy írhatjuk a property-n keresztül.
Visible	A mező láthatóságát állíthatjuk be vele.

A Table komponens

A Table komponens használata a legegyszerűbb és leggyorsabb módja egy adatbázis egy táblájának elérésének. A komponens gyakorlatilag az adatbázis egy fizikai tábláját reprezentálja. A Table komponens a TdataSet osztály leszármazottja, de számos olyan új tulajdonsággal és metódussal rendelkezik, amivel az ős nem.
Ahhoz, hogy egy táblához kapcsolódjunk, a következő dolgokat kell tennünk:
Be kell állítani a DatabaseName tulajdonságban az adatbázis aliasát, vagy lokális adatbázisok esetén annak az elérési útját. A TableName tulajdonságban be kell állítani annak a táblának a nevét, amelyet a komponens reprezentál az adatbázisból.
Az Active property-t True-ra kell állítani.

Fontosabb tulajdonságok:

Exclusive	Az adatbázis zárolása, ha True akkor másik alkalmazás nem férhet hozzá a tábla adataihoz.
IndexDefs	Információt tartalmaz a tábla indexeiről.
IndexFieldCount	Az adott indexkulcsban lévő mezők számát adja.
IndexFieldNames	Az indexben szereplő mezők felsorolása
IndexFields	Tfield típusú tömb, egy index mezőiről tartalmaz információkat.
IndexName	Egy létező index nevének megadására szolgál. (az itt beállított index szerinti lesz a rendezettség)
MasterFields	Master-detail kapcsolatban itt kell beállítani a kapcsolatot felépítő mezőket.
MasterSource	Master-detail kapcsolatban, ha ez a tábla egy segédtábla, akkor ebben a property-ben kell beállítani azt az adatforrást, ami a főtáblára van állítva.
ReadOnly	Ha True, a tábla nem szerkeszthető, csak olvasható.
TableName	A fizikai tábla neve, amit a komponens reprezentál.
TableType	Tábla típusa (pl. Paradox, dBASE).

Fontosabb metódusok:

AddIndex	Új indexet hoz létre a táblához.
BatchMove	Rekordok másolása egy másik táblából ebbe a táblába.
CreateTable	Tábla létrehozása.
DeleteIndex	Töröl egy indexet.
DeleteTable	Törli a táblát.
EmptyTable	Törli a tábla összes rekordját
GetIndexNames	Lista az adathalmaz indexeiről.
GotoKey	Megadott kulcs szerinti rekordra mozgatja a rekordmutatót.
GotoNearest	A kurzort a kulcshoz legközelebbi rekordra mozgatja.
LockTable	A tábla zárolása.
RenameTable	Tábla átnevezése.
SetKey	Engedélyezi kulcs beállítását az adathalmazhoz.
UnlockTable	Megszünteti a tábla zárolását.

Master-detail kapcsolat (fő, és segéd adatok)

Tegyük fel, hogy van egy megrendeléseket tartalmazó táblánk, amelynek egy-egy rekordja tartalmaz egy rendelési azonosítót, a teljesítés dátumát, a rendelés összértékét, a megrendelő nevét stb. Egy megrendelésnek ugyebár vannak tételei is, ezeket érdemes egy külön táblában eltárolni. Ennek a külön táblának egy-egy rekordja tartalmazza egy adott árucikk adatait, és azt a rendelési azonosítót, ami a fő táblában is megtalálható, ugyanis ezen azonosító alapján tudunk kapcsolatot létesíteni a két tábla között. Tehát a segédtáblában annyiszor szerepel a rendelési azonosító, ahány különböző árucikket rendelünk. Ez egy a többhöz kapcsolat a két tábla között. A master-detail kapcsolatban gyakorlatilag a főtáblából vett azonosító alapján szűrjük a melléktáblát, és csak a feltételnek megfelelő rekordok jelennek meg belőle. Ahhoz, hogy a kapcsolatot létre tudjuk hozni, a segédtábla kapcsolódó mezőjének indexeltnek kell lennie.

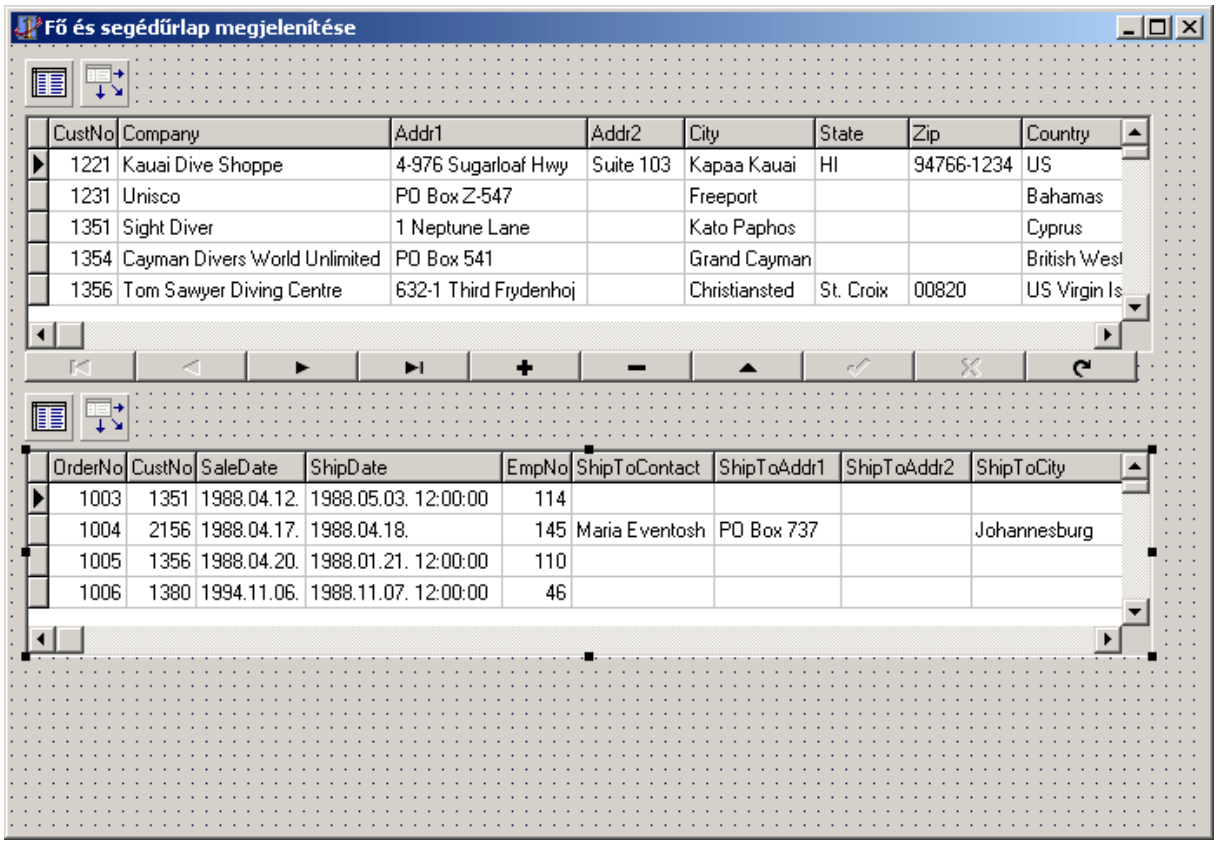
Feladat: Fő és segédűrlap (FOSEGED):

A most következő példában az egyik tábla az ügyfeleinket, a másik a hozzájuk tartozó rendeléseket tartalmazza. Példánkban ismét segítségül hívjuk a Delphi példaadatbázisait. Hozzuk létre a munkához a FOSEGED könyvtárat, majd kezdjünk egy új projectet.

1. Form Name: frmFoseged, Caption: Fő és segédűrlap megjelenítése. Save All: Ufoseged, Pfoseged.
2. Tegyünk egy Table komponenst a Form-ra és állítsuk be a következő tulajdonságokat: Name: MasterTable, DatabaseName: DBDEMOS, TableName: customer.db, Active: True
3. Tegyünk egy DataSource komponenst a Formra, neve legyen MasterSource, a Dataset pedig MasterTable.
4. Most egy DBGrid, és egy DBNavigator komponens következik (a Data Controls palettán találjuk őket), a DataSource property legyen a MasterSource.
5. Tegyünk egy újabb Table komponenst a Form-ra a következő tulajdonságokkal: Name: DetailTable, DatabaseName: DBDEMOS, TableName: orders.db, Active: True

6. Újabb Datasource komponens következik, a neve legyen DetailSource, a DataSet pedig DetailTable.
7. Még egy DBGrid komponens van hátra, amit állítsunk a DetailTable-ra. (a DataSource property-t állítsuk a DetaliSource komponensre)

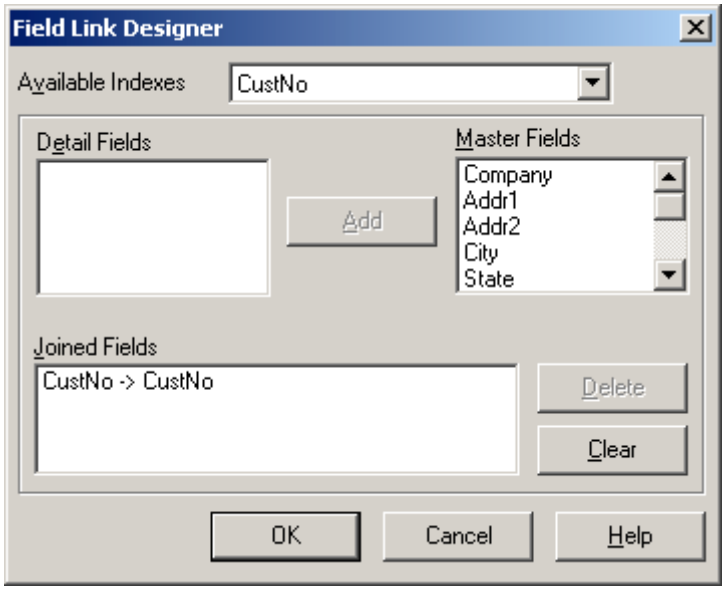
Ha mindent a fentiek szerint hajtottunk végre, valami hasonlót kellene látnunk tervező nézetben:



Láthatóan megjelennek az adataink már tervezés alatt is, de jelenleg még semmiféle kapcsolat nincsen a két tábla között, így adataik egymástól teljesen függetlenül jelennek meg. Nyilván a felső tábla adatai alapján szeretnénk, ha az alsó táblánkban a fent kiválasztott főadathoz tartozó adatok jelennének csak meg. A kapcsolatot mindig a segédtábla felől kezdeményezzük:

8. Ehhez, a DetailTable komponensen kattintsunk a MasterSource propety-re és állítsuk be a MasterSource adatforrás komponenst.
9. Ha ez megvan, még a MasterField property beállítására van szükség. Kattintsunk a MasterField tulajdonságra, ekkor bejön a Field Link Designer.
10. Itt az Available Indexes legördülő menüben válasszuk a CustNo indexet (hiszen a CustNo lesz a kapcsolódó mező). Ezután a Detail Fields és a Master Fields listában válasszuk a CustNo mezőket, majd nyomjuk meg az Add gombot.:

Amennyiben megfelelően dolgoztunk, ezt kellene látnunk:



Ekkor a Joined Fields ablakban már látható is a két mező közötti kapcsolat. Ezt követően Ok gombot nyomva, máris létrejött a kapcsolat a két tábla között. Ha mentettünk, és futtatunk, látni fogjuk, hogy a második DBGrid-ben a második oszlopban (CustNo) csak az első Grid-ben aktív elemek fognak megjelenni, melyeket az egérrel, a kurzormozgató billentyűvel, vagy akár az általunk elhelyezett DBNavigátorral is vezérelhetünk.

A Data Modul

A Data Modul egy speciális form amire szerkesztési időben rápakolhatjuk az adatelérési és egyéb nem vizuális komponenseket. A Data Modul futási időben nem jelenik meg. Létrehozni a File/New/DataModule menüben tudjuk. Segítségével az adatbázis-elérési komponenseket a különböző formoktól elkülönítve tudjuk tárolni, ezáltal megvalósul az adatelérési logika és a felhasználói felület elkülönítése. Ez annál is inkább fontos, mert így egy összetett adatbázis esetében elegendő, ha egyetlen felületen (az adatmodulon) hozzuk létre a szükséges elemeket, majd hivatkozunk rájuk a kezelőfelületen.

Felhasznált adatmegjelenítési komponensek

Az adatmegjelenítési komponensekről már korábban volt szó, de részletes tárgyalásukra csak később kerül sor. Feladatuk az adatelérési komponensek által szolgáltatott adatok megjelenítése. Itt a részletezéstől eltekintve, most csak a példa programban felhasznált adatmegjelenítési komponensekről esik néhány szó.

DBGrid — Adathalmaz rekordjainak táblázatos formában történő megjelenítésére szolgál. Számos property-je közül a DataSource a legfontosabb, ezt kell arra az adatforrás komponensre állítani, amely által szolgáltatott adatokat meg kívánjuk jeleníteni.

DBNavigator — Navigációs gombokat tartalmazó vizuális komponens, mellyel egy adathalmazban mozoghatunk, illetve lehetőségünk lesz új rekord felvételére, törlésére, módosítására.

DBEdit — Egy rekord egy mezőjének megjelenítésére, editálására szolgál. A DataSource tulajdonságban adjuk meg az adatforrást, a DataField property-ben pedig a mező nevét.

DBText — Egy rekord egy mezőjének megjelenítésére szolgál. A DataSource tulajdonságban adjuk meg az adatforrást, a DataField propertyben pedig a mező nevét.

A Database Desktop

A Database Desktop egy Delphi-hez adott segédprogram, mellyel helyi adatbázistáblákat tallózhatunk, dBase vagy paradox táblákat hozhatunk létre, indexeket definiálhatunk, tábla struktúrát módosíthatunk stb. Ezt a felületet indíthatjuk közvetlenül a Delphi-ből is (Tools/Database Desktop), de a telepítés után elérhető a Startmenü/Programok/Borland Delphi5/Database Desktop útvonalon is.
Ezek után lássunk hozzá az első önálló (nem a példaadatbázison alapuló) példa programhoz, ami egy egyszerű telefonszám nyilvántartó program lesz. Egyszerűségéből adódóan mindössze egy táblán fog alapulni, így a táblák közötti relációkra nem ad példát, azt majd egy másik példaprogramon keresztül nézzük meg.

Feladat: Önálló telefonszám nyilvántartó (TELEFON):

Először hozzuk létre az adatbázistáblát. Ehhez indítsuk el a Database Desktop-ot, és válasszuk a New/Table menüpontot., ezen belűl pedig a dBASE IV tábla formátumot. Az itt megjelenő ablakban tudjuk megadni a meződefiníciókat.

A következő mezőket hozzuk létre:

Field Name	Type	Size	Dec
VEZ_NEV	C	20	
KER_NEV	C	20	
IRSZAM	C	4	
VAROS	C	20	
CÍM	C	30	
TELEFON	C	11	
EMAIL	C	30	
EGYEB	C	30	

Ezek után hozzuk létre az indexeket a következő mezőköz:

VEZ_NEV
KER_NEV
IRSZAM
VAROS
TELEFON

Ehhez nyomjuk meg a jobb felső részen, a lenyíló menü alatt lévő Define gombot (A lenyíló menü tartalma "Indexes" kell hogy legyen, ha nem az akkor állítsuk arra). Bejön egy új ablak, ahol a mezők közül ki lehet választani azt, amelyikhez indexet szeretnénk készíteni. Válasszunk ki egy mezőnevet (klikk a mezőn), majd nyomjunk OK gombot. Az ezután bejövő ablakban egy felajánlott nevet találunk az indexünknek, ez alapértelmezetten a mező neve. Ezt fogadjuk el. Ha minden indexszel készen vagyunk, mentsük el a táblánkat abba könyvtárba, ahova majd a programunkat is lementjük, vagyis hozzuk létre a Delphi könyvtárunkban a TELEFON alkönyvtárat, táblánkat pedig nevezzük „phone.dbf”-nek.
Hogy a táblát könnyen elérhessük, definiálunk egy aliast az adatbázisunkhoz. Ehhez a DataBase Desktop Tools menüjében indítsuk el az Alias Managert. Keressük meg a New gombot, majd rákattintva állítsuk be a következőket:
A Database alias legyen „phone”, a típusa maradjon STANDARD, az elérési útnál pedig állítsuk be az előbb létrehozott TELEFON alkönyvtárat. (Nagy segítségünkre lesz a jobb oldalon látható Browse… gomb. Ha készen vagyunk, nyomjuk meg a Keep New gombot, majd az OK-ra kattintva elkészültünk az alias létrehozásával. Közben meg fog jelenni egy kérdés, mely arra vonatkozik, hogy az általunk végrehajtott változtatásokat beírhatja-e a Delphi az Idapi.cfg állományba. Természetesen válaszoljunk igennel, végül zárjuk be a Database Desktop-ot.

Miután megvagyunk a táblával és az aliassal, nekiállhatunk a program elkészítésének.

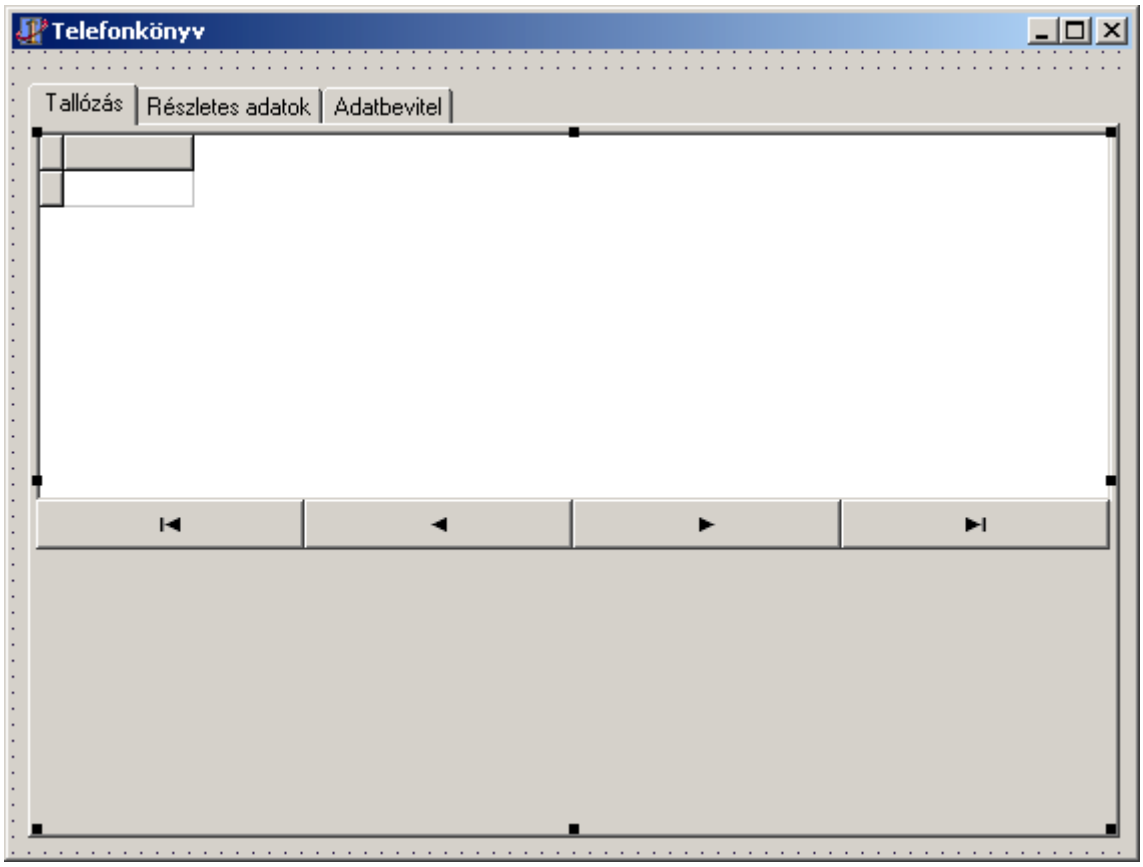
1. Kezdjünk egy új projectet (File/New Allplication), majd mentsük el a TELEFON könyvtárba ahova phone.dbf-et mentettük a unitot Uphone néven, a projectet pedig Pphone néven.
2. Adjunk egy DataModul-t a projecthez (File/New/Data Module). A neve legyen DM (Object Inspector Name), és mentsük le DMUnit néven (Save All).
3. Tegyünk egy Table és egy DataSource komponenst a DataModulra a következő beállításokkal:

Table	
Tulajdonság	*Érték*
Database Name	Phone
Name	Phone
TableName	Phone.dbf
Active	True

DataSource	
Tulajdonság	*Érték*
AutoEdit	False
DataSet	Phone
Name	PhoneDS

Ha a Table komponensnél nem állítjuk be a DatabaseName tulajdonságot, a program a könyvtárában lévő táblákkal dolgozik. Ezért fontos, hogy a project és a táblánk egyazon könyvtárban legyenek.
Az Active property True-ra váltásával már szerkesztési időben is láthatjuk egy adatbázistábla tartalmát. (illetve csak láthatnánk, ha nem lenne üres)
A DataSource komponens DataSet tulajdonságában kell beállítani azt az adathalmazt, aminek tartalmát a DataSource majd szolgáltatni fogja. Jelen esetben ez a Phone, ami egyébként most nem is lehetne más, hiszen csak egy adathalmazunk van. Az AutoEdit False-ra állítása arra szolgál, hogy egy adatmegjelenítési komponensben csak akkor tudjunk szerkeszteni, ha először az adathalmazt dsEdit módba billentettük az Edit metódus segítségével. (AutoEdit=True esetén ez automatikusan megtörténik, ha egy adatbevitelre alkalmas adatmegjelenítési komponensben lenyomunk egy billentyűt)
Most jöhet a felhasználói felület kialakítása. Programunk egyetlen formból fog állni, amin egy PageControl komponens segítségével külön oldalakon jelenítjük meg a program különböző funkcióit.

4. A Form1 tulajdonságait kérjük vissza, a Name frmTelefon, a Caption: Telefonkönyv legyen.
5. Helyezzünk el egy PageConrol elemet (Win32 paletta) PageControl-on hozzunk létre három lapot (jobb egér gombbal kérhetünk New Page-et, majd az Object Inspectorban szerkeszthetjük tulajdonságait). Az első lapon tallózhatunk ismerőseink telefonszámai között, a második lapon a részletes adatokat jeleníthetünk meg, a harmadikon pedig adatokat vihetünk fel. (Ennek megfelelően állítsuk be a Caption tulajdonságokat "Tallózás", "Részletes Adatok" és "Adatfelvitel" -re).
6. Készítsük el a tallózás felületét. Ehhez vegyünk fel egy DBGrid, és egy DBNavigator elemet a „Tallózás” lapunkra. Helyezzük el őket az ábrán látható módon:



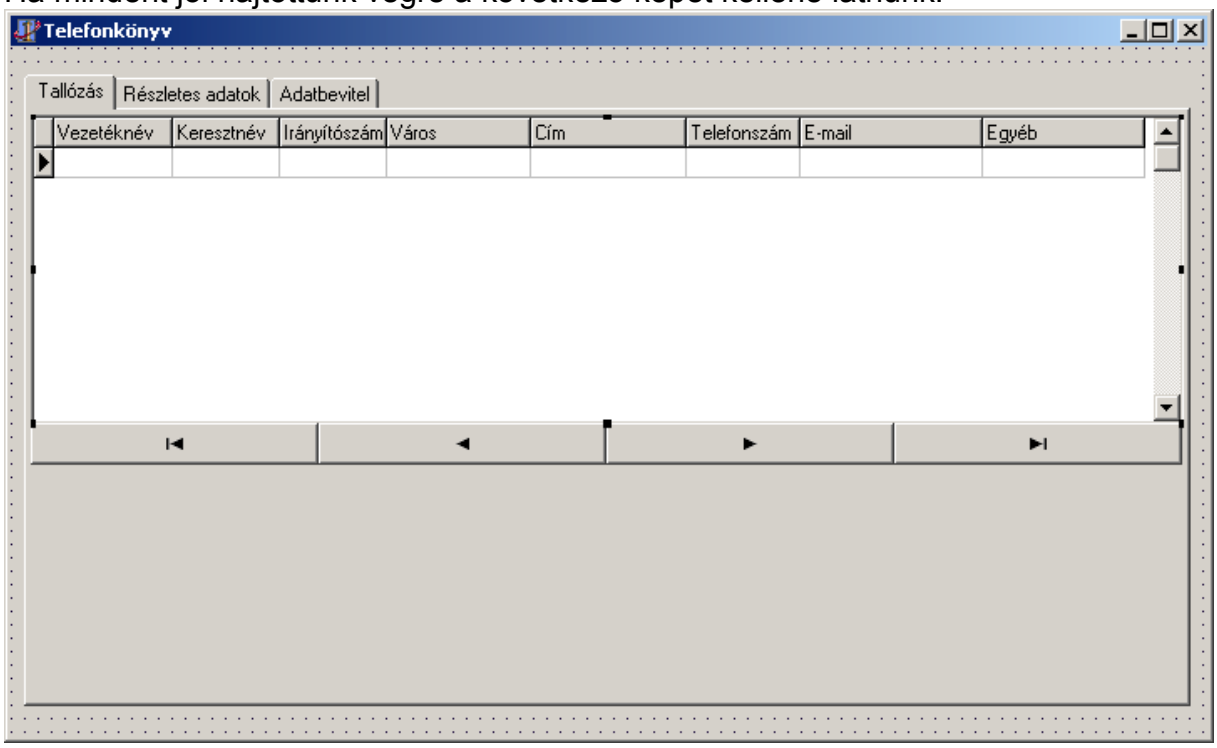
Mint látható, a DBNavigator már csupán a léptető gombokat tartalmazza, amit a Navigátor VisibleButtons tulajdonságánál állíthatunk be (csak az első 4 elem legyen True, a többit tiltsuk le: False). Ha megpróbálnánk beállítani a háttérben lévő adatmodulba felvett táblát a Grid DataSource-hoz, azt tapasztaljuk, hogy az inaktív, vagyis nem tudjuk kiválasztani. Ennek oka a korábban már tapasztalt hivatkozásokban keresendő: azért nem látjuk az adatforrást, mert azt a DataModul-ra tettük. Ahhoz hogy el tudjuk érni a hőn áhított PhoneDS adatforrásunkat, mindössze a form unit-jának *Implementation részébe be kell írnunk a Uses DMUnit; sort. Ezek után meg kell hogy jelenjen a DM.PhoneDS a lenyíló* menüben. Tehát ha egy DataModul-on lévő adatforrást el akarunk érni egy másik form-ról, akkor először a form számára "láthatóvá" kell tennünk azt, az előbb leírt módon.

7. Lépjünk át a szerkesztőablakba, keressük meg a Uphone Unitunkat, majd a USES részben hivatkozzunk az adatmodulra:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, DBCtrls, Grids, DBGrids, ComCtrls, DMUnit;
```

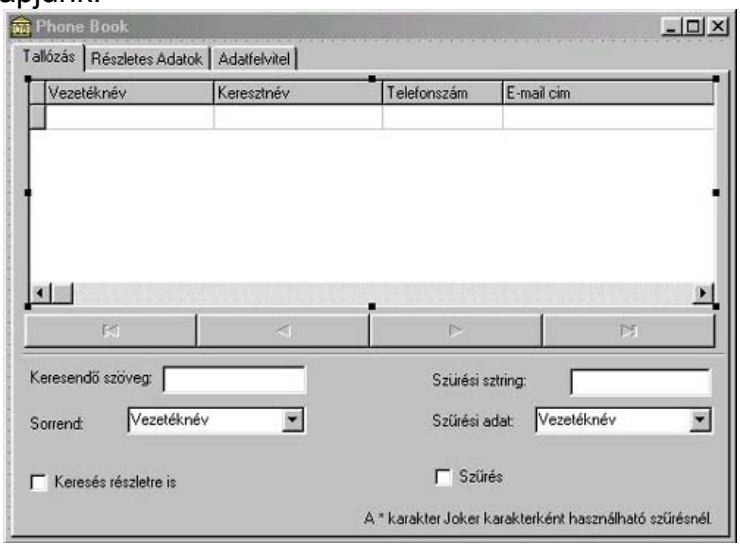
Ezt követően, ha mentettünk, és visszatértünk a formunkhoz, már használni tudjuk a hivatkozást. Állítsuk be tehát a Grid forrásának a DM.PhoneDS-t, és meg fognak jelenni az áhított mezők. Természetesen a méreteket módosítva igyekezzünk úgy elrendezni a Grid-et, hogy valamennyi adat láthatóvá váljon. Kérdés még az oszlopok felirata, hisz az most az általunk adott neveket tartalmazza, de a megjelenítésnél ez nem túl stílusos. Talán emlékszünk még a mezőszerkesztőre, lépjünk át az adatmodulra, kattintsunk duplán a táblánkra, adjuk hozzá az összes mezőt, majd a mezőkre kattintva egyesével állítsuk be a DisplayLabel tulajdonságot.
Az irányítószám és a telefonszám esetén az EditMask tulajdonságban megadhatjuk, hogy csak számokat fogadjon el, illetve a telefonszámnál még a számokat elválasztó

szeparátort is beállíthatunk. A Required property True-ra állításával azt érjük el, hogy annak a mezőnek, ahol ez be van állítva mindenképpen értéket kell adnunk, ha egy új rekordot viszünk fel, különben a rendszer nem engedi a rekord felvitelét. A VEZ_NEV, KER_NEV és TELEFON mezőknél állítsuk a Required tulajdonságot True-ra. Tehát minimálisan ez a három adat minden személyről be kell hogy kerüljön az adatbázisunkba, másképpen nem tudjuk felvinni egy adatát sem. Állítsuk be a tulajdonságokat, majd térjünk vissza a formunkra.
Ha mindent jól hajtottunk végre a következő képet kellene látnunk:



Segítsük még a felhasználót néhány keresési, és szűrési mezővel:

8. Helyezzünk el két Edit-et, két ComboBox-ot, két CheckBox-ot, egy Bevel-t és néhány Label-t a formunk alsó szabad területére. (Vegyük észre, hogy nem foglalkozunk a DB elemekkel, mindezek a standard palettán vannak) Helyezzük el úgy a komponenseket, hogy egy a következő képen látható felületet kapjunk:



Az elemek alapértelmezett tulajdonságain ne változtassunk, maradjon minden ahogy elhelyeztük. Készítsük most el a sorba rendezést megvalósító részt.

9. A táblánk mindig a bal oldali ComboBox-ban megjelenő tulajdonság szerinti rendezettségben jelenjen meg. Ehhez a ComboBox.Text property-je legyen Vezetéknév.
10. Lépjünk át az adatmodulra, majd a Table komponens IndexName property-jét állítsuk VEZ_NEV-re . Ezek az apalértelmezett beállítások lesznek, tehát amíg a felhasználó nem állít be új rendezési sorrendet, a VEZ_NEV index szerinti sorrend lesz az alapértelmezett.
11. A ComboBox Items tulajdonságát töltsük fel a következő értékekkel:

    Vezetéknév
    Keresztnév
    Irányítószám
    Város
    Telefon

    Ezek azok a mezők, amelyek szerint még rendezni lehet majd a táblát.
12. A ComboBox OnChange eseményébe írjuk a következő kódot:

```
procedure TfrmTelefon.ComboBox1Change(Sender: TObject);
begin
 case ComboBox1.Itemindex of
    0: DM.Phone.IndexName:='VEZ_NEV';
    1: DM.Phone.IndexName:='KER_NEV';
    2: DM.Phone.IndexName:='IRSZAM';
    4: DM.Phone.IndexName:='VAROS';
    5: DM.Phone.IndexName:='TELEFON';
 end;
ComboBox3.ItemIndex:=ComboBox1.ItemIndex;
end;
```

Ennek hatására a kiválasztott érték szerint lesz indexelve a tábla, így a sorrend is ilyen lesz. (A ComboBox3 egy másik lapon lévő lenyíló menü, így a két ComboBox tartalma együtt változik majd)

A következő feladatunk a keresést segítő mező kialakítása:

13. A keresendő szöveget egy egyszerű Edit komponensbe kell beírni. A keresés akkor indul, ha Entert nyomunk, ezért a kis kereső kódunkat az Edit1 komponens OnKeyDown eseményébe írjuk:

```
procedure TfrmTelefon.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
If Key = VK_RETURN Then
   If not CheckBox1.Checked Then
     Begin
     If DM.Phone.FindKey([Edit1.Text]) Then
        Edit1.SelectAll
     End
   Else
     Begin
     DM.Phone.FindNearest([Edit1.Text]);
     Edit1.SelectAll;
     End;
end;
```

A FindKey, az aktuális Index szerinti mezőben keres a megadott értékre, a FindNearest szintén az aktuális index szerinti mezőben hozzávetőleges keresést valósít meg. A CheckBox1-el állítjuk be, hogy csak pontos találatot fogadjon el (FindKey), vagy szótöredékre is keressen (FindNearest). Találat esetén az Edit-be beírt szöveget kijelöltté tesszük, így ha újabb szóra akarunk keresni, nem kell először kitörölni a beírt tartalmat, hanem egyből tudjuk írni a következő keresési feltételt. Maradt még valami, ez pedig a szűrés.

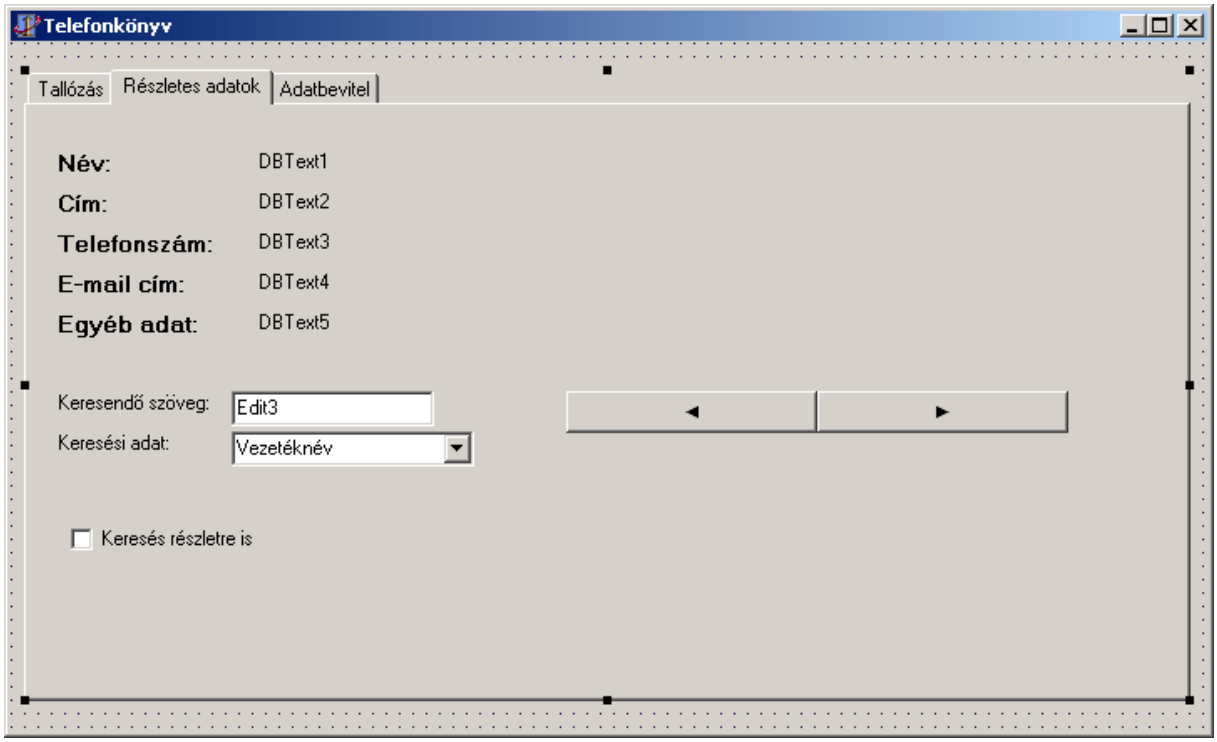
14. A szűrés segítségével a megjelenített adatok számát szűkíthetjük le adott feltétel szerint. Pl. csak a budapesti ismerősöket jelenítjük meg, vagy pl. csak azokat, akiknek Szabó a vezetékneve. A ComboBox tulajdonságait ugyanúgy állítsuk be ahogy a keresésnél, majd a Szűrést aktiváló CheckBox2 OnClick eseményjellemzőjébe írjuk be:

```
procedure TfrmTelefon.CheckBox2Click(Sender: TObject);
begin
  With DM.Phone do Begin
        Case ComboBox2.ItemIndex of
           -1: Filter:='VEZ_NEV='''+Edit2.text+'''';
           0: Filter:='VEZ_NEV='''+Edit2.text+'''';
           1: Filter:='KER_NEV='''+Edit2.text+'''';
           2: Filter:='IRSZAM='''+Edit2.text+'''';
           3: Filter:='VAROS='''+Edit2.text+'''';
           4: Filter:='TELEFON='''+Edit2.text+'''';
        End;
        Filtered:=CheckBox2.Checked;
  End;
end;
```

A szűrési feltételt a Table komponens Filter property-ébe kell beírni, és a Filtered property True-ra állításával aktiválhatjuk a szűrést. Látható, hogy a ComboBox által kiválasztott mező-t tesszük egyenlővé az Edit2 komponensbe beírt értékkel, és ez lesz a szűrési feltételünk. Tehát ha pl. a Város-t választjuk, és az Edit komponensbe beírjuk hogy Budapest és a Szűrést bekapcsoljuk, akkor csak a Budapesti személyek jelennek meg a rácsban. Ezzel tulajdonképpen elkészítettük az első oldalunkat, jöjjön a következő lap, a „Részletes adatok” elkészítése.

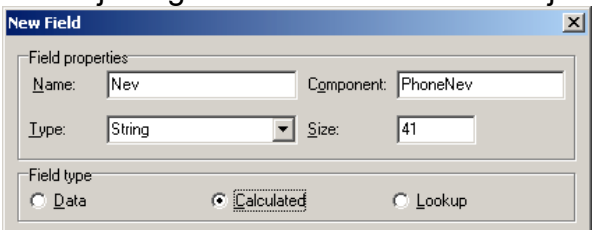
Ebben az ablakban tekintheti majd meg a felhasználó, a telefonkönyv rekordjait egyenként, amit még kiegészítünk két számított mezővel a gyakorlás kedvéért.

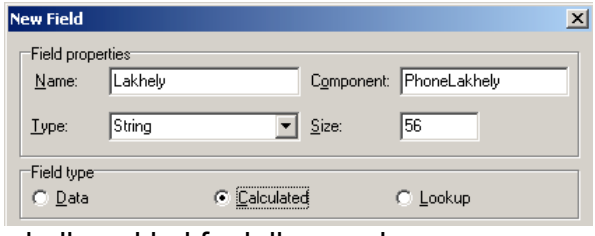
15. Helyezzük el a következő komponenseket a második lapon, a képen látható módon:



A félkövér „Név:”, „Cím:”, stb mezők egyszerű Label-ek (Standard paletta). Mellettük helyeztük el a képen látható DBText-eket (Data Controls paletta), alattuk van egy Label, és egy szimpla Edit (Standard), ezek alatt egy Label, és a harmadik ComboBox (Standard), melyre az előző oldali kódunkban hivatkoztunk is. Legalul szerepel a CheckBox3 (Standard), és jobb oldalon a DBNavigator (Data Controls), mely csupán előre-hátra léptetést enged (VisibleButtons közül csupán a nbPrior, nbNext True, a többi False). Érdekesség a Név, és a Cím, ezek lesznek a számított mezőink.

16. Lépjünk át az adatmodulunkra (DM), keressük meg a táblánkat, kattintsunk rá a jobb gombbal, válasszuk a Fields Editort, a megjelenő ablakban pedig a jobb gombbal New Fields-et kérjünk. Állítsuk be a következőket:





A mezők alsó része inaktív lesz, így nem is kell azokkal foglalkoznunk.

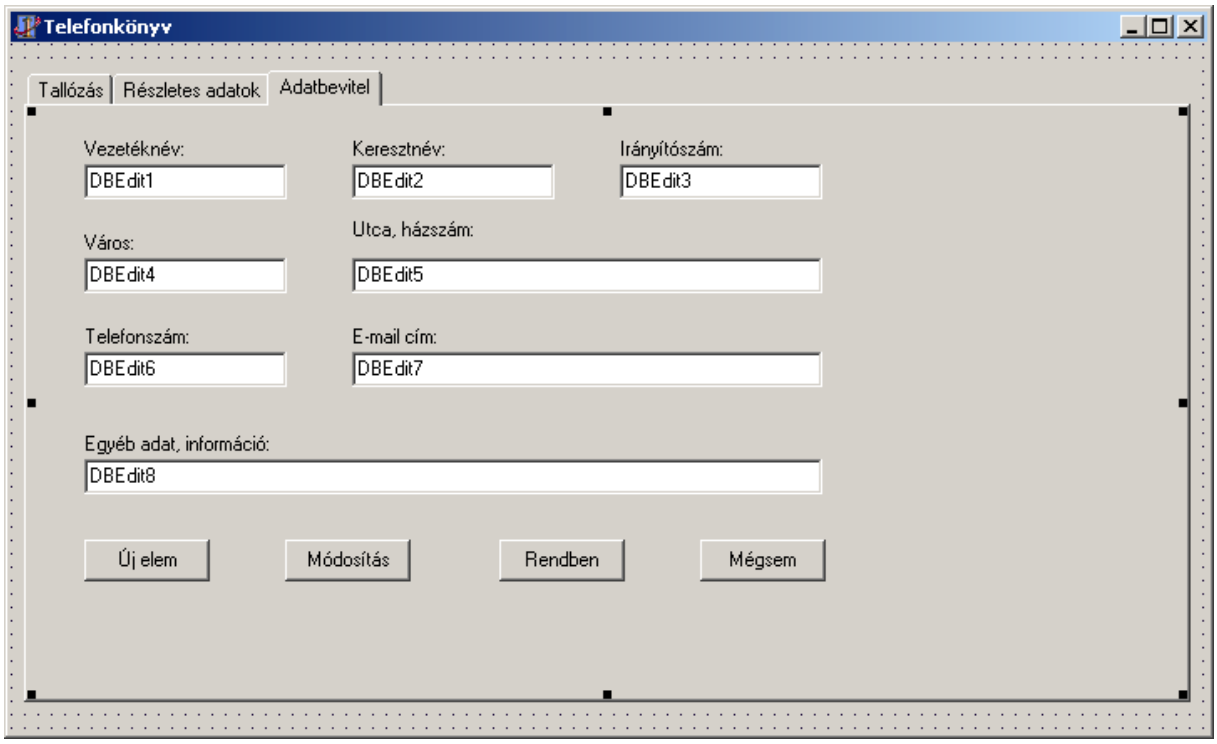
Nos, ezzel létrehoztuk a két számított mezőt, de ez még kevés ahhoz, hogy egy adott rekord esetén ezekben a mezőkben megjelenjen a várt érték. Ez csak akkor történik meg, ha a Table komponens (Phone) OnCalcFields eseményjellemzőjébe beírjuk a mező értékét előállító kódot.

17. Zárjuk be a DM.Phone mezőlistát tartalmazó ablakát, majd lépjünk a táblánk Events fülecskéjére, és keressük meg az onCalcFields eseményt. A kódszerkesztőbe pedig írjuk be a következő kódot:

```
procedure TDM.PhoneCalcFields(DataSet: TDataSet);
begin
 With Phone do Begin
        FieldByName('Nev').AsString:=
        FieldByName('VEZ_NEV').AsString +' '+ FieldByName('KER_NEV').AsString;
        FieldByName('Lakhely').AsString:= FieldByName('IRSZAM').AsString +' '+
        FieldByName('VAROS').AsString +', '+FieldByName('CIM').AsString;
 End;
end;
```

Ezzel el is készültünk, térjünk vissza a „Részletes adatok” lapunkhoz, majd rendeljük hozzá a megfelelő mezőneveket a DBText-ekhez.

18. A DBText1 mező DataSource értéke legyen a DM.phoneDS, a DataField pedig (értelemszerűen) a most létrehozott Nev mező. Állítsuk be valamennyi elemünk értékét a megfelelő helyekre. Ne lepődjünk meg, ha eltűnnek a feliratok, hiszen nem tartalmaz adatot a hivatkozott táblánk. Ideje erről is gondoskodnunk, vagyis tegyük lehetővé az adatfelvitelt.
19. Lépjünk az utolsó fülre („Adatbevitel”), majd helyezzük el a következő elemeket a lapon:



A feliratok itt is egyszerű Label-ek, a mezők azonban most kivétel nélkül DBEdit-ek (Data Controls paletta). És csak érdekességként: ha nem tetszenek a Navigátor gombjai, gyárthatunk mi magunk is gombokat (Button, a Standard palettáról).

20. A DBEdit-ek tulajdonságainál a DataSource, és a DataField állítása szükséges az előzőekben már leírt módon. Ha lenne adat a táblánkban, itt már látnánk is azt.
21. A gomboknál a táblához való hozzáférést fogjuk tudni befolyásolni. Az első gombunk („Új elem") a bővítésért lesz felelős. A gomb tulajdonságainál az Events fülön az OnClick eseményhez írjuk be a következő kódot:

```
procedure TfrmTelefon.Button1Click(Sender: TObject);
begin
  DBEdit1.SetFocus;
  DM.Phone.Append;
end;
```

DBEdit1.SetFocus mindössze annyit tesz, hogy a DBEdit1 komponensre állítja a fókuszt, bárhol is volt eddig, hiszen valószínűleg sorban akarjuk majd felvinni az adatokat.

22. Következő gombunk a „Módosítás", kódja pedig:

```
procedure TfrmTelefon.Button2Click(Sender: TObject);
begin
  DBEdit1.SetFocus;
  DM.Phone.Edit;
end;
```

Ebben az esetben szerkeszteni szeretnénk az aktuális adatot.

23. Az előző két gomb hozzáférést biztosított a táblában lévő adatokhoz, a „Mégsem" gombunk ezt fogja „visszavonni", vagyis ezzel tudjuk mind a létrehozás (új rekord), mind pedig a szerkesztés műveletét érvényteleníteni. A kódunk:

```
procedure TfrmTelefon.Button4Click(Sender: TObject);
begin
  DM.Phone.Cancel;
end;
```

24. Utolsó gombunk pedig az előző műveletek jóváhagyása lesz majd, vagyis a felvitel, és a módosítás végrehajtása:

```
procedure TfrmTelefon.Button3Click(Sender: TObject);
begin
  If DM.Phone.State in [dsEdit,dsInsert] Then
  DM.Phone.Post;
end;
```

Kódunkban először megvizsgáljuk, hogy az adathalmaz dsEdit (szerkesztés) vagy dsInsert (beszúrás) állapotban van-e, és ha igen, akkor meghívjuk a Post metódust. Hogyha ezt a vizsgálatot nem végeznénk el, akkor hibaüzenetet kapnánk minden olyan esetben, mikor úgy hívjuk meg a Post metódust, hogy az adathalmaz nincs a szerkesztési állapotok egyikében sem.

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Mire alkalmas a mezőobjektumok EditFormat tulajdonsága?
   a., Bemeneti maszk megadására
   b., Szerkesztéskori formátum kijelzésre
   c., Formázás engedélyezésére
2. Mire alkalmas egy táblánál a MasterSource tulajdonság?
   a., Főtábla kapcsolására
   b., Segédtábla kapcsolására
   c., Adatforrás beállítására
3. Miért nem jelenik meg közvetlenül az általunk létrehozott adatmodul az alkalmazásunk űrlapjain?
   a., Mert önálló Unitban van
   b., Mert önálló űrlapon van
   c., Mert mindenhez nekünk kell forrást rendelnünk
4. A FindNearest segítségével?
   a., pontos találatra kereshetünk
   b., szótöredékre kereshetünk
   c., szavakra kereshetünk

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. A FindKey az azonosítók keresését teszi lehetővé.
   igaz
   hamis
2. Az adatmodulok az adatbázisnak csak adatmodell szintű (táblák, mezők, kapcsolatok) megjelenítésének lehetőségeivel rendelkeznek.
   igaz
   hamis
3. Egy Table komponens használatához annak legalább 3 tulajdonságát be kell állítanunk.
   igaz
   hamis
4. Ha egy mezőre beállítjuk a Requied tulajdonságot, akkor kötelezővé tesszük annak kitöltését.
   igaz
   hamis
5. Azt, hogy egy mezőhöz milyen mezőobjektum tartozik, a mező típusa határozza meg.
   igaz
   hamis

Adatbáziskezelés III.

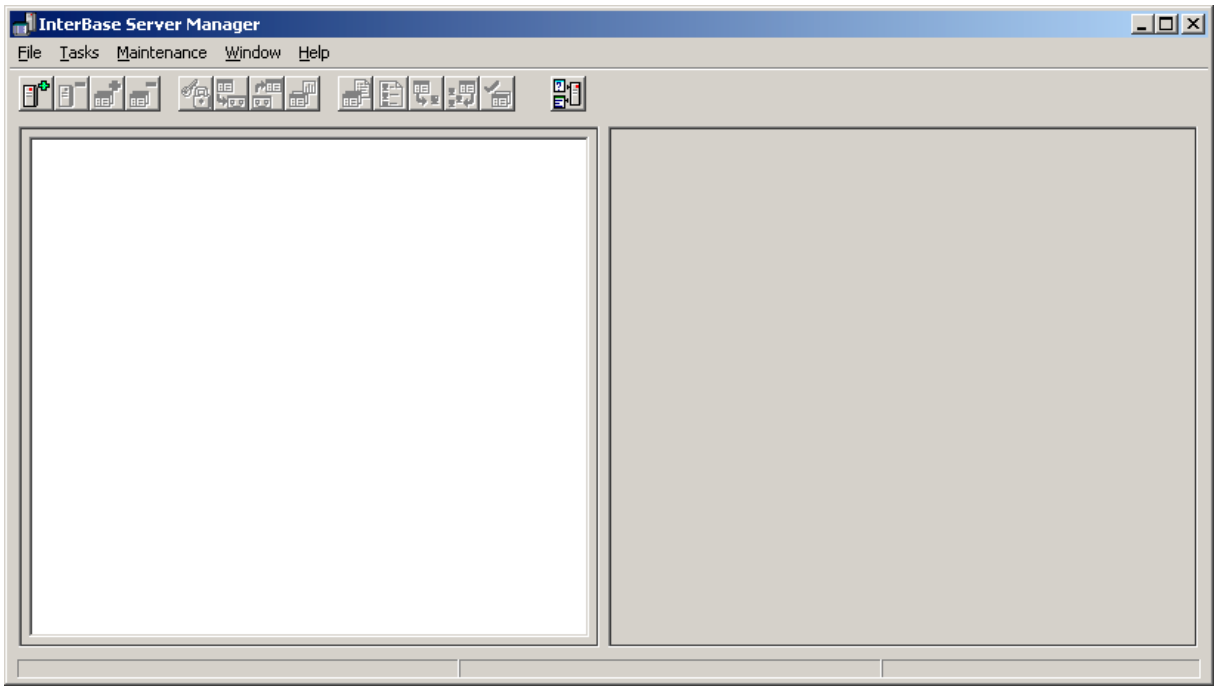
Az előző fejezetekben igyekeztünk megalapozni az adatbázis kezeléssel kapcsolatos ismereteket, ebben a fejezetben egy kicsit tovább ismerkednénk a Delphi lehetőségeivel. Többször szerepelt már példaként a tanárokat, és tantárgyaikat nyilvántartó egyszerű 3 táblás adatmodell, így annak részletes magyarázatától e fejezetben eltekintünk. A példánkban teljesen nulláról fogjuk létrehozni az adatbázist, Interbase, illetve Dbase felülettel.

Feladat: Tanárok és tárgyaik adatbázis (TANTARGY):

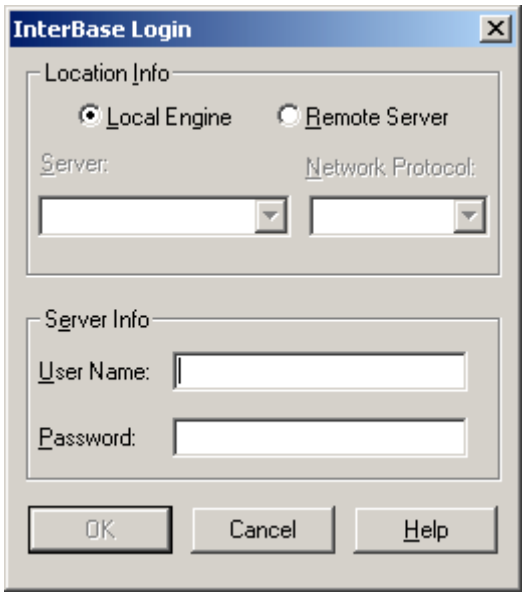
Hozzuk létre a fenti könyvtárat a Delphi alkönyvtárban. Mivel az előző alkalommal foglalkoztunk a DBase IV táblák létrehozását segítő Database Desktoppal, így most az Interbase segítségével készítjük el az adatokat tároló táblákat.

Interbase adatbázis készítése:

1. Ha a telepítés során feltelepítettük az Interbase Server-t, akkor a Start menüből most el tudjuk azt érni a Start/Programok/Interbase/Server manager útvonalon (feltéve, ha nem változtattuk meg a telepítés során az alapértelmezett beállításokat.) indítsuk el a Server managert. A következő kép fogad majd bennünket:

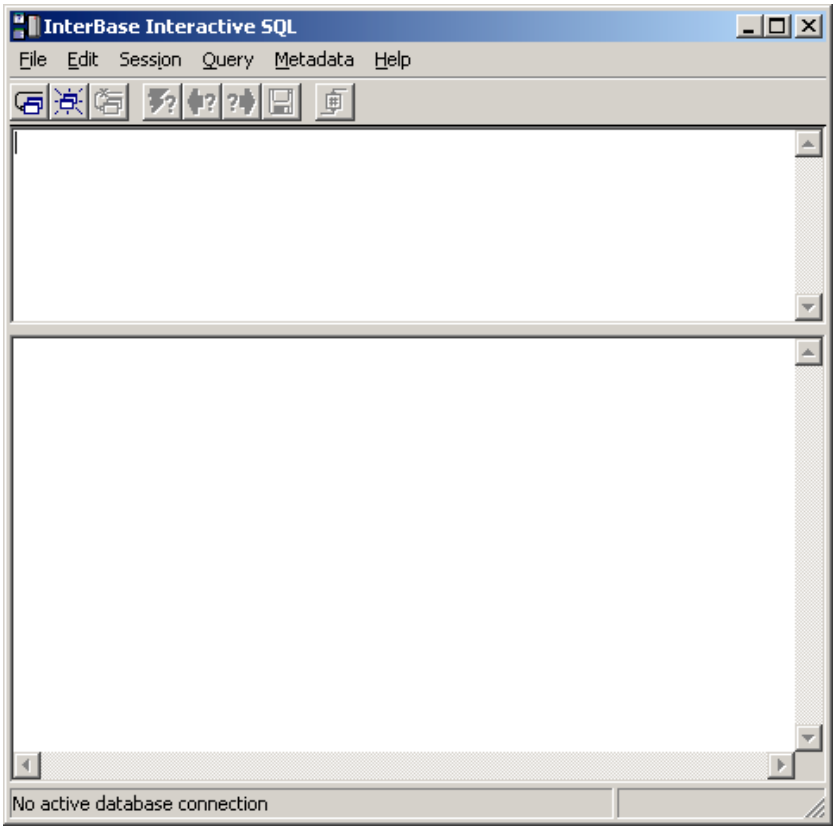


2. Mint látható, gyakorlatilag sok választási lehetőségünk nincsen, be kell jelentkeznünk a Serverre, hogy bármilyen tevékenységet végre tudjunk hajtani (bár a bennünket leginkább érdeklő Interactive SQL elérhető!). Válasszuk tehát a Server login lehetőséget a File menü, vagy az eszköztáron lévő gomb segítségével. Amennyiben megfelelően telepítettünk el kell tudnunk érni a Local Engine-t:



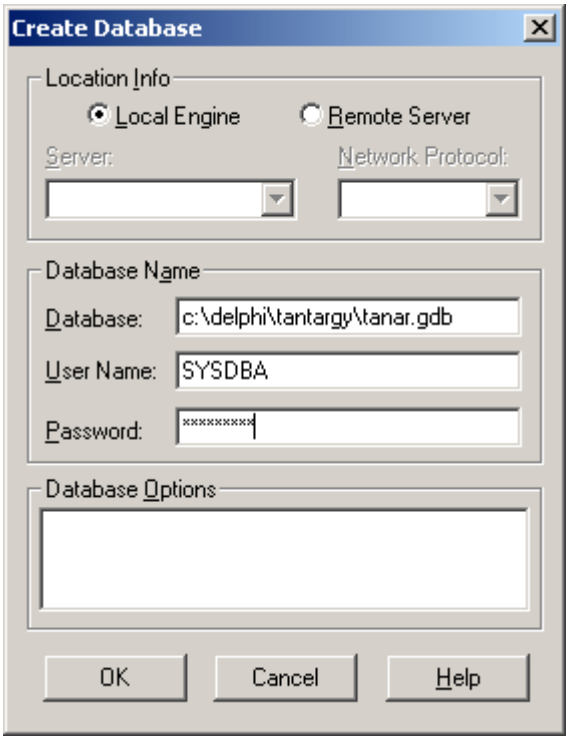
Ha mégis inaktív a helyi szerver, akkor csak távoli gépre tudnánk bejelentkezni, ami elég jelentős mértékben megnehezítené az adatbázis létrehozását. A lokális szerver használatához az Interbase egy beépített, állandó rendszergazdai szintű felhasználót definiált, melynek felhasználói neve: „SYSDBA”, jelszava pedig: „masterkey”. Adjuk meg az adatokat, majd hagyjuk jóvá az „OK” gombbal. (Case Sensitive!) ezzel sikerült is a helyi Serverre történő bejelentkezésünk. Ha figyelmesen körülnézünk, látni fogjuk, hogy a főablak nem teszi lehetővé új adatbázis létrehozását. Indítsuk most el a beépített interaktív SQL-t:

3. Válasszuk a Task/Interactive SQL lehetőségét, vagy az eszköztárunk utolsó gombját. A következő ablakot fogjuk látni:



Az ablak felső részébe kell gépelnünk a végrehajtandó SQL utasításokat, majd az utasítás végrehajtását az eszköztáron lévő „Execute Query” gomb megnyomásával kezdeményezhetjük. Amennyiben hibát követtünk volna el, úgy ezt jelezni fogja a gép egy hibaüzenettel, melyben pontosan meg tudjuk majd nézni a hiba okát is. Ha jó volt az utasításunk, akkor hibaüzenet nélkül le fog futni a kód, majd az alsó ablakban megjelenik a korábban begépelt SQL kódunk. (Ez hiba esetén is megtörténik, de akkor az utasítás nem kerül végrehajtásra!).
Jelenleg nincsen adatbázisunk, létre kellene hoznunk azt:

4. Lépjünk a File menübe, majd válasszuk a Create Database lehetőséget. A megjelenő ablakban adjuk meg a szükséges adatokat, végül hagyjuk jóvá a bevitelt:

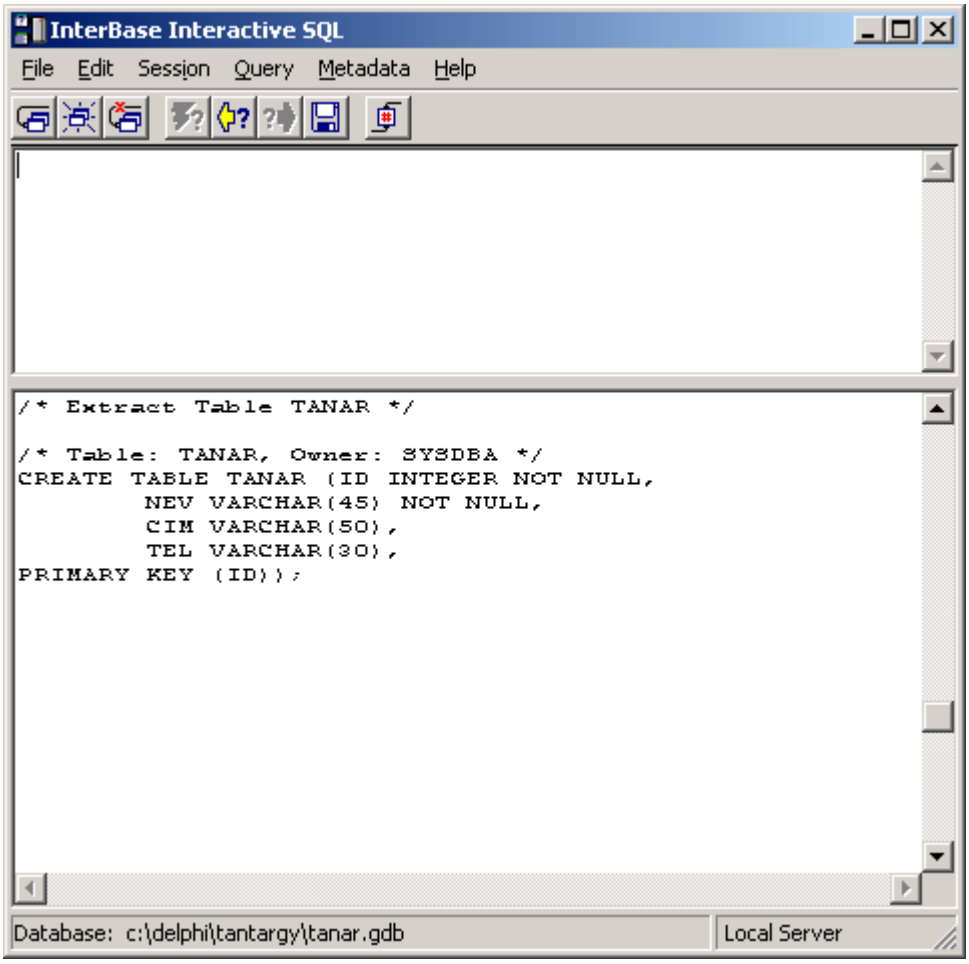


Mint látható, az adatbázis létrehozásakor a teljes elérési útvonalat adtuk meg, végül pedig az állomány neve a „tanar.gdb”. Természetesen nem lenne szerencsés, ha Önök is a „C” meghajtó gyökerére hivatkoznának, az elérési út legyen a saját meghajtójukon lévő delphi könyvtár.
Amennyiben minden rendben, az ablak alsó részében meg fog jelenni az elérési útvonal, és az intézőben látni is fogjuk a tanar.gdb-t. ellenőrizzük, majd folytassuk tovább a munkát.
Az Interbase GDB állományáról annyit érdemes rögtön tudnunk, hogy egyetlen állományban fog megjelenni valamennyi táblánk, és a köztük definiált kapcsolatunk, indexeink, stb. Ahhoz, hogy első táblánkat létrehozzuk, kapcsolódnunk kell az adatbázishoz, de ezt a gép (a lent látható elérési útvonallal jelezve) már megtette helyettünk. Ha később szeretnénk még majd valamit végrehajtani az adatbázissal kapcsolatban, akkor már nem kell létrehoznunk, csupán csatlakoznunk kell hozzá a File Connect To Database utasításával. Hozzuk tehát létre első táblánkat:

5. Az interaktív SQL felső részébe írjuk be a következő kódot:

```
Create Table TANAR
(ID Integer Not Null,
NEV Varchar(45) Not null,
CIM varchar(50),
TEL varchar(30),
Primary key(ID)
);
```

Ezt követően kattintsunk az Execute Query gombra, majd ha mindent rendben hajtottunk végre megjelenik az alsó ablakban a kódunk, a felső ablak pedig üressé válik. Nyilván elbizonytalanodhatunk, hogy valóban létrejött-e a kívánt tábla, amit ellenőrizhetünk a Metadata menü Extract Table menüből. Ekkor megkérdezi majd a gép, hogy elmentse-e egy állományba az információt, itt válaszoljunk majd „Nem”-et. Kérdésünkre (ha minden jól megy) a következő képpel fog válaszolni a gép:



Hozzuk létre a további két táblánkat:

```
Create Table TARGY
(ID integer not null,
MEGNEVEZES varchar(30),
LEIRAS varchar(50),
primary key(ID)
);
```

6. Majd jöhet a szak táblánk, mely a kapcsolótábla szerepét tölti be. Ne feledkezzünk meg a kapcsolatok definiálásáról:

```
create table SZAK
(TANARID integer not null,
TARGYID integer not null,
foreign key(TANARID) references TANAR(ID),
foreign key(TARGYID) references TARGY(ID)
);
```

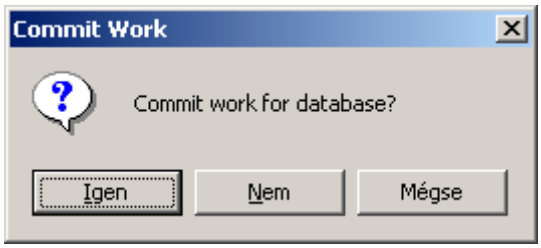
Amennyiben biztosak szeretnénk lenni abban, hogy minden rendben van, természetesen menet közben is ellenőrizhetjük a felvitel helyességét. Lehetőségünk van azonban a felvitel végén egy átfogó ellenőrzésre is: Metadata/Extract Database. A kérdésre válaszoljunk megint nem-et. Ekkor az alsó ablakban a teljes adatbázisunk kódja meg fog jelenni. Mivel az SQL modulban az SQL utasításokat már megismerték, így nem tértünk ki jelen jegyzetben az utasítások teljes körű magyarázatára. Inkább emlékeztetés (ismétlés) képpen, vegyünk fel a táblákba egy-egy rekordot. (Bár ez nyilván sokkal egyszerűbb lesz majd a Delphiben elkészítendő vizuális kezelőfelületen…)

7. Gépeljük még be a következő SQL utasításokat:

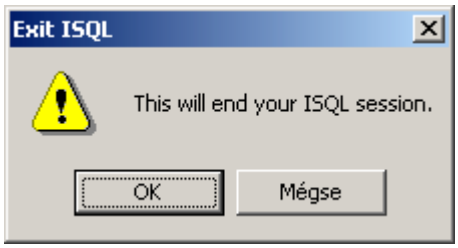
```
insert into TANAR values(1,'Kis Pál','Budapest','06-30-123-4567');
insert into TARGY values(1,'Matematika','A számok világa');
insert into SZAK values(1,1);
```

Ezzel el is készítettük első önálló Interbase alapú adatbázisunkat. Hagyjuk tehát el a szervert:

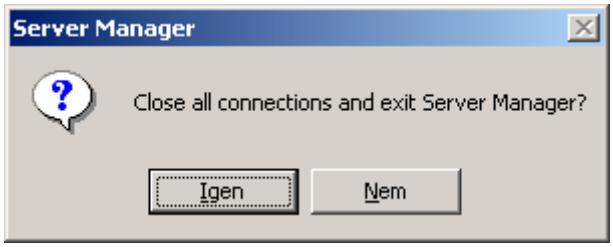
8. Zárjuk be az Interaktív SQL ablakát az „x” mentén. A kérdésre válaszoljunk igen-el.



9. A megjelenő figyelmeztetést hagyjuk jóvá (OK):



10. Végül zárjuk be a Server Manager ablakát is, ahol ismét egy üzenetet kapunk:

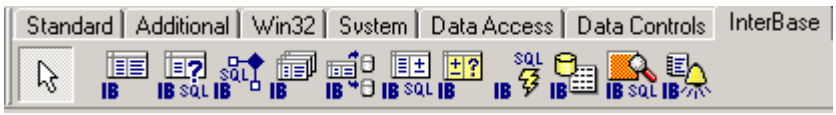


Válaszoljunk itt is igen-el.
Kezdhetjük a fejlesztést, nem maradt más hátra, mint a felület kialakítása, melyet a Delphiben készítünk majd el. Zárjuk be a Database desktop-ot, indítsuk el a Delphit, és végezzük el a következő tevékenységeket:

11. A megjelenő form Name: frmFo, Caption: Interbase adatbázis kezelés.
12. Save All: Ufo, Pfo a korábban létrehozott TANTARGY könyvtárba.

Az Interbase adatbázisokkal való műveletekhez a delphi külön palettát kínál, nézzük meg ennek tartalmát:

Az Interbase komponenspaletta:



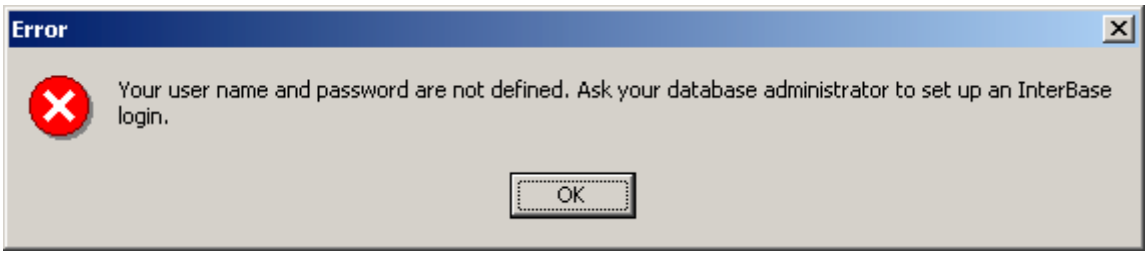
IBTable: Interbase tábla megjelenítése
IBQuery: Interbase lekérdezés
IBStoredProc: Tárolt eljárás hívása
IBDatabase: Kapcsolódás interbase adatbázishoz
IBTransaction: tranzakciók végrehajtása adatbázisok között
IBUpdateSQL: csak olvasható lekérdezések frissítése
IBDataSet: SQL lekérdezéseink megjelenítéséhez
IBSQL:Közvetlen SQL utasítás végrehajtása
IBDatabaseInfo: A kapcsolt adatbázis információinak elérése
IBSQLMonitor: Dinamikus SQL megjelenítéséhez
IBEvents: Események regisztrálása, követése.

Mivel az Interbase felület esetén más az adatbázis struktúra, így ennek a felületnek az eléréséhez más jellegű vezérlők, komponensek szükségesek. Építsük most fel adatbázisunk egyszerű kapcsolatát, és kezelő felületét.

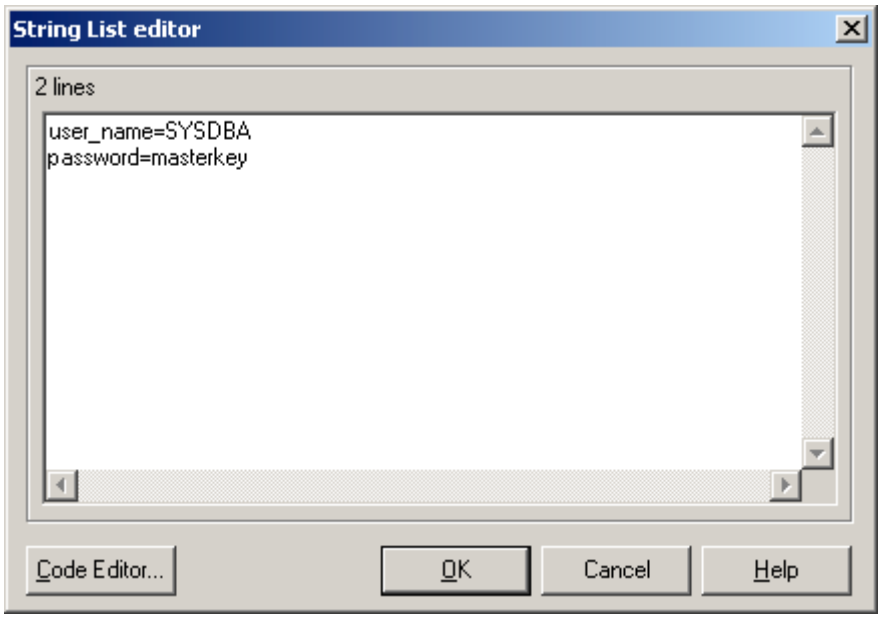
13. Helyezzünk el egy IBDatabase komponenst a formunkra. A DataBaseName tulajdonság kiválasztásánál az alapértelmezett könyvtárban szereplő adatbázis rögtön elérhető számunkra, így válasszuk ki azt. Ha aktiválni szeretnénk a hozzáférést az adatbázishoz (Active: True), be fogja kérni a rendszer a felhasználói név, jelszó párost. Adjuk meg a SYSDBA masterkey adatokat, majd aktívvá válik az adatbázis. Jelen állapotban már meg tudnánk tekinteni az adatokat, de ahhoz hogy fel is tudjunk vinni adatot, szükségünk lesz egy tranzakció kezelő komponensre is, melyet a táblánkhoz kellene kapcsolni.
14. Helyezzünk el egy IBTransaction-t a formra, majd kapcsoljuk azt az adatbázisunkhoz. Állítsuk be a DefaultDatabase tulajdonságot (IBDatabase1),

majd az Active tulajdonságot is True-ra. Ha most mentünk, és futtatunk, bár semmi megjeleníthető adatunk nincsen, de a kapcsolatot felépítettük, így elvileg működik is minden. Futásidőben azonban (majd látni fogjuk), megint meg kell adnunk a hozzáféréshez szükséges adatokat, bár a felhasználói név már ki van töltve. Ez elég kellemetlen, kínos lenne, ha minden felhasználónak el kellene árulnunk a rendszergazdai jelszót, így tegyük hozzáférhetővé oly módon az adatbázisunkat, hogy ne kérjen jelszót.

15. Ehhez keressük meg az adatbázis elemünket, majd a tulajdonságainál találunk egy LoginPrompt tulajdonságot. Ha ezt egyszerűen megpróbáljuk bekapcsolni, az aktív tulajdonság True-ra állításakor kapunk egy hibaüzenetet:

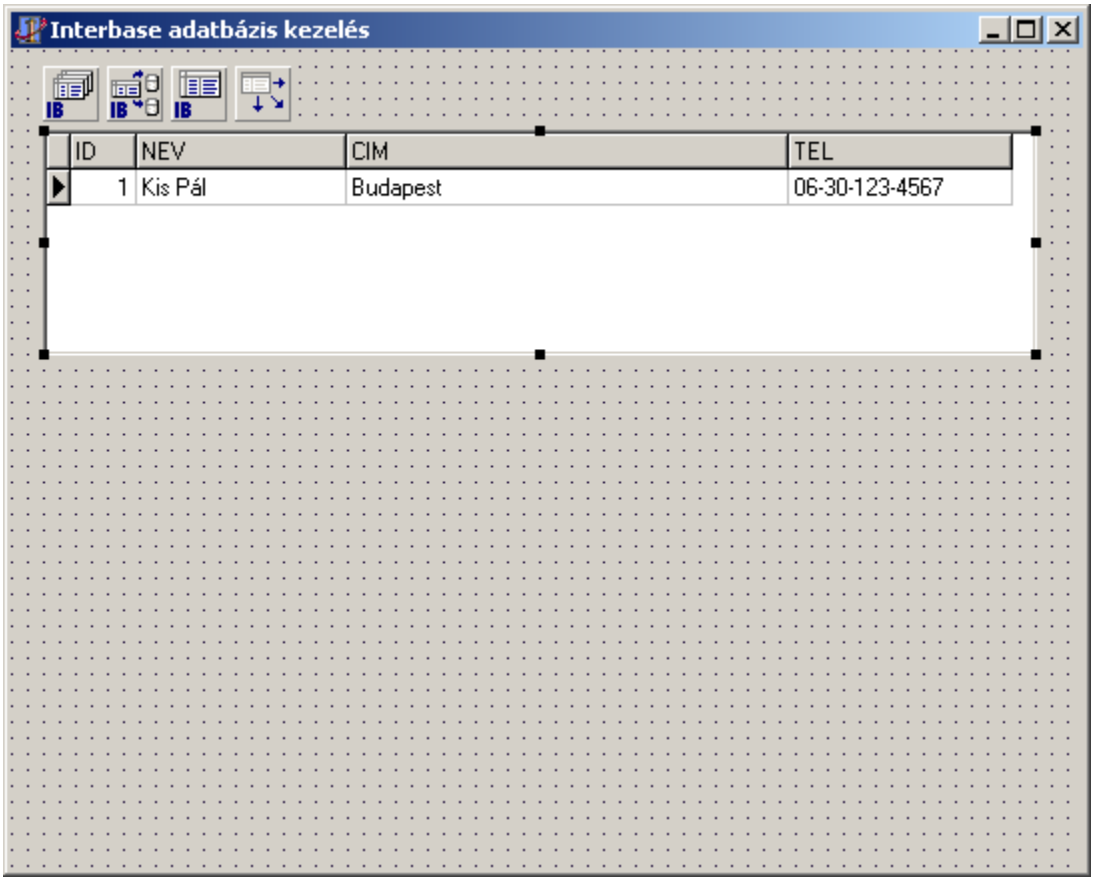


Az ok viszonylag egyszerű: meg kellene adnunk legalább nekünk egyszer statikusan a jogosultságunkat az adatbázishoz, ahhoz, hogy ezt követően állandóan hozzáférhessünk. Nézzük meg most az adatbázis Params tulajdonságát, és csaljuk elő azt a … -ra kattintva.

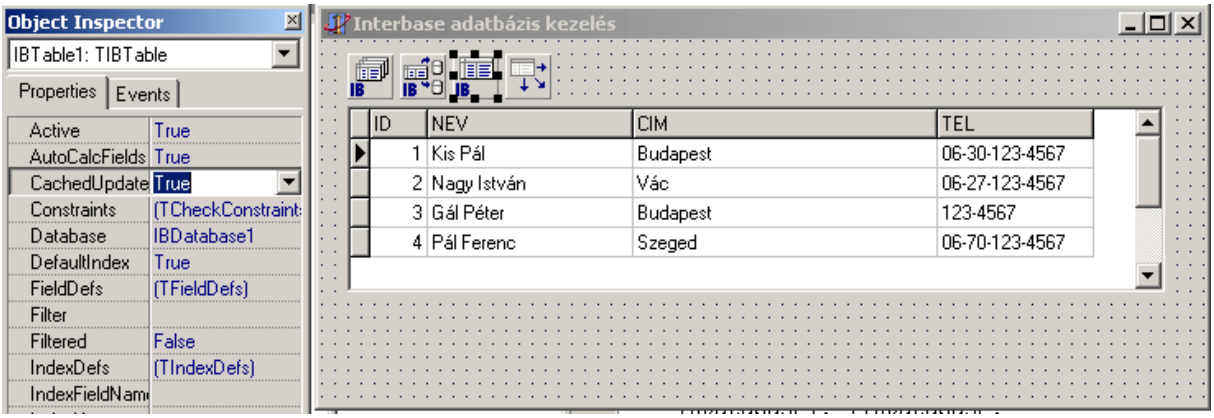


Itt tudjuk paraméterezni a belépést, legalább itt szerepelnie kell a jelszónak ahhoz, hogy többet ne reklamáljon a rendszer a hozzáférésnél. Írjuk be a fenti paramétert, majd a LoginPrompt mostmár lehet False, végül állítsuk True-ra az aktív tulajdonságot. Ha most mentünk, és futtatunk, már nem fogunk hibaüzenetet kapni, sőt már jelszó sem szükséges az eléréshez. Jelenítsük meg most az adatokat:

16. Helyezzünk egy IBTable komponenst a formra, állítsuk be a tulajdonságait: DatabaseName: IBDatabase1, TableName: TANAR, Active: True.
17. a DataAccess palettáról helyezzünk el egy DataSource elemet, melyet irányítsunk az IBTable1-re. (DataSet: IBTable1).
18. Jöhet egy DBGrid (DataControls paletta), melynek DataSource-a legyen a DataSource1, és már látnunk is kell az egyetlen felvitt tanárunk adatait:



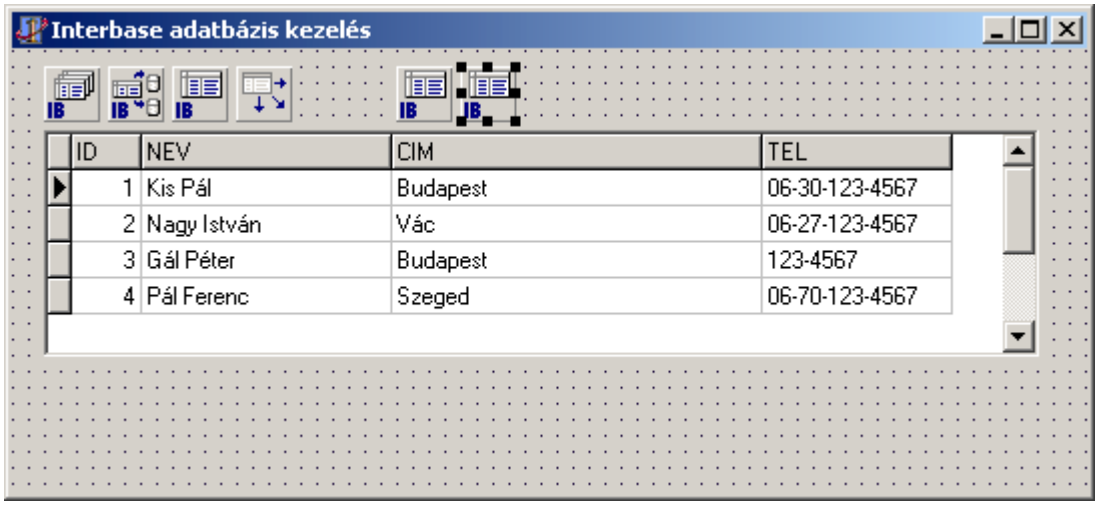
Most mentsünk, majd futtassunk, és győződjünk meg róla, hogy minden rendben. Úgy tűnik igen, de ha felviszünk futásidőben egy új rekordot, akkor az nem jelenik meg a táblában, ha visszatérünk tervező nézetbe. De ez a kisebbik baj, újra futtatva programunkat, már futásidőben sem látható a felvitt új rekord. Lépjünk most ki a projectből (File menü Close All), és ha kéri a gép, akkor mentsünk mindent. Ha ezt követően a File menü Reopen utasításával újra előhívjuk a projectet (az első helyen lesz a listánkban), akkor azonnal ott lesz a felvitt adat. Tehát a művelet végrehajtásra került (ezt jelezte, hogy nem kaptunk hibaüzenetet), de a megjelenítőnk adatai nem frissültek. Ennek oka a táblánk CachedUpdate tulajdonságának False értéke. Módosítsuk azt, majd ezt követően meg fognak jelenni a frissen felvitt adataink is.



Töltsük fel a táblát a fent látható adatokkal, majd kérjünk mentést.
Nyilvánvalóan az Interbase komponensekkel hasonló módon dolgozhatunk, mint a korábban már megismert elemekkel, vagyis a kapcsolatokat figyelembe véve tudjuk

felépíteni kezelő felületünket. Egy egyszerű módszer segítségével itt oldjuk még meg a három táblához való közvetlen hozzáférést, mégpedig egyetlen Grid segítségével.

19. Helyezzünk el még két IBTable komponenset a Grid felett, és mindkét táblánál állítsuk be a Database :IBDatabase1, és a TableName tulajdonságot. Az első táblánk mutassa a tárgyak adatait, a második pedig a szak tábla adatait. Ne feledkezzünk meg a táblák CachedUpdate tulajdonságának True-ra állításáról.



20. Helyezzünk el a Grid alatt három gombot, melyek a Grid-ben megjelenő adatok változtatásáért lesznek felelősek. Bal oldalon legyen a btnTanar, Caption: Tanárok. Középen, btnTargy, Caption: Tárgyak, jobb oldalon pedig btnSzak, Caption: Szakok.
21. Kattintsunk duplán a tanárok gombján, majd a kódszerkesztőbe írjuk be a következő kódot:

```
procedure TfrmFo.btnTanarClick(Sender: TObject);
begin
DataSource1.DataSet:=IBTable1;
end;
```

Természetesen kódunknak egyenlőre nagy hatása nem lesz, hisz az eredeti forrásunk is a tanárok tábla volt. Írjuk meg tehát a többi tábla csatolásáért felelős kódokat is.

22. Kattintsunk a tárgyak gombunkra, majd jöhet a kód:

```
procedure TfrmFo.btnTargyClick(Sender: TObject);
begin
DataSource1.DataSet:=IBTable2;
end;
```

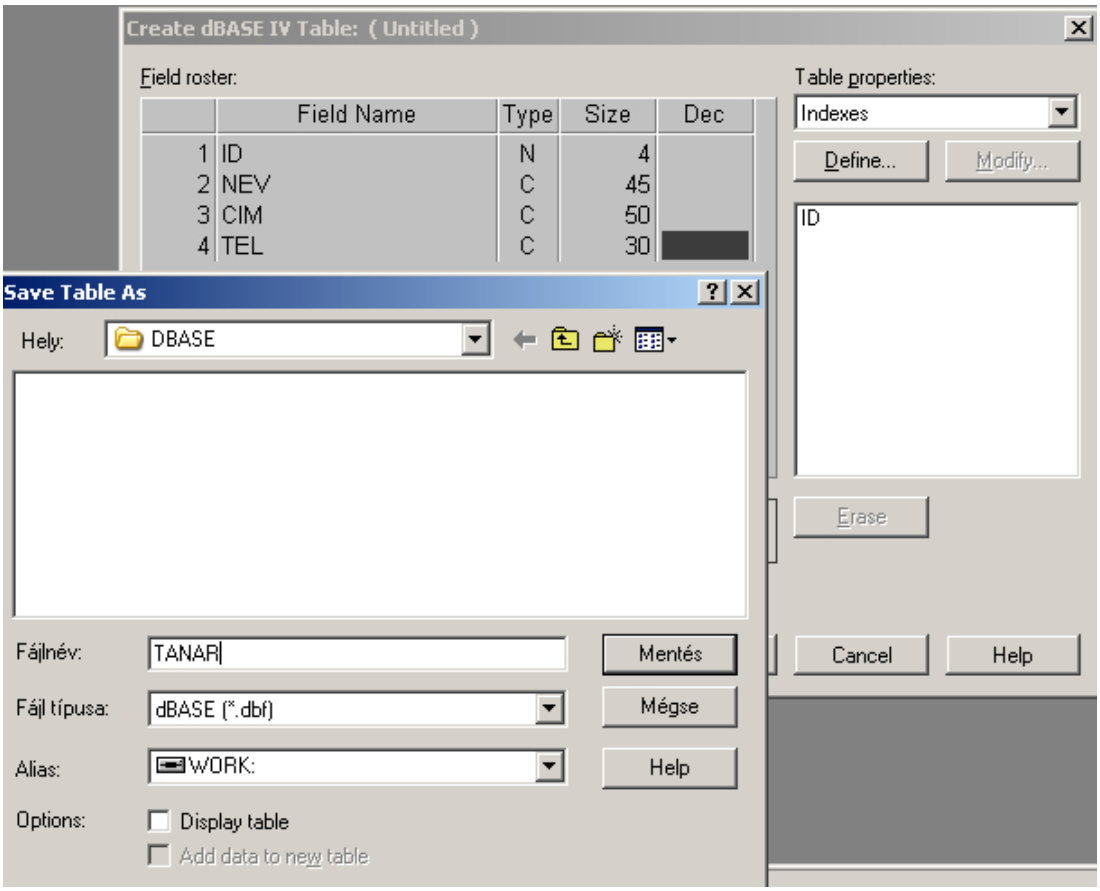
23. Végül a harmadik táblánkat is rendeljük az adatforráshoz önállóan, majd mentsünk, és futtassunk. Ha mindent rendben találtunk, el is készült első Interbase alapú felületünk alapja.

A feladatban igyekeztünk nulláról elkészíteni az adatbázisunkat. Látható, hogy a komponensek futásidőben történő átirányításával viszonylag egyszerűvé tehető egy bonyolultnak tűnő műveletsor is. Ha még elhelyeznénk egy egyszerű menüt a táblák hozzárendelésével, készítenénk néhány lekérdezést SQL-ben, valamint jelentést, teljes értékű (bár bonyolultnak nem nevezhető) adatbázishoz jutnánk. Nézzük mindezt Dbase-ben.

Tanárok és tantárgyaik adatbázis DBASE-ben (DBASE):

Hozzuk létre a szükséges könyvtárat (DBASE), majd indítsuk is el a Delphit.

1. A form neve: frmFo, Caption: Dbase adatbázis a tanárokról, és tárgyaikról.
2. Save All: Ufo, Pfo a létrehozott DBASE könyvtárba.
3. Menjünk a Delphi Tools menüjének Database Desktop almenüjébe, majd az „adatbázis asztalon" válasszuk ki a File menü Working Directory… lehetőségét. Itt adhatjuk meg a munkakönyvtárat, vagyis azt, hogy a tevékenységeinkhez közvetlenül hová kínálja fel a gép a mentést. Állítsuk be az előzőleg létrehozott DBASE könyvtárunkat, majd hagyjuk jóvá a beállítást.
4. Vegyük fel az első (TANAR) táblánkat a korábban leírt mezőkkel együtt, definiáljunk egy indexet a Tanar táblánk ID mezőjére, majd mentsük azt a munkakönyvtárunkba. Ha felvittük az adatokat, és a mentésnél tartunk, a következőt kellene látnunk:

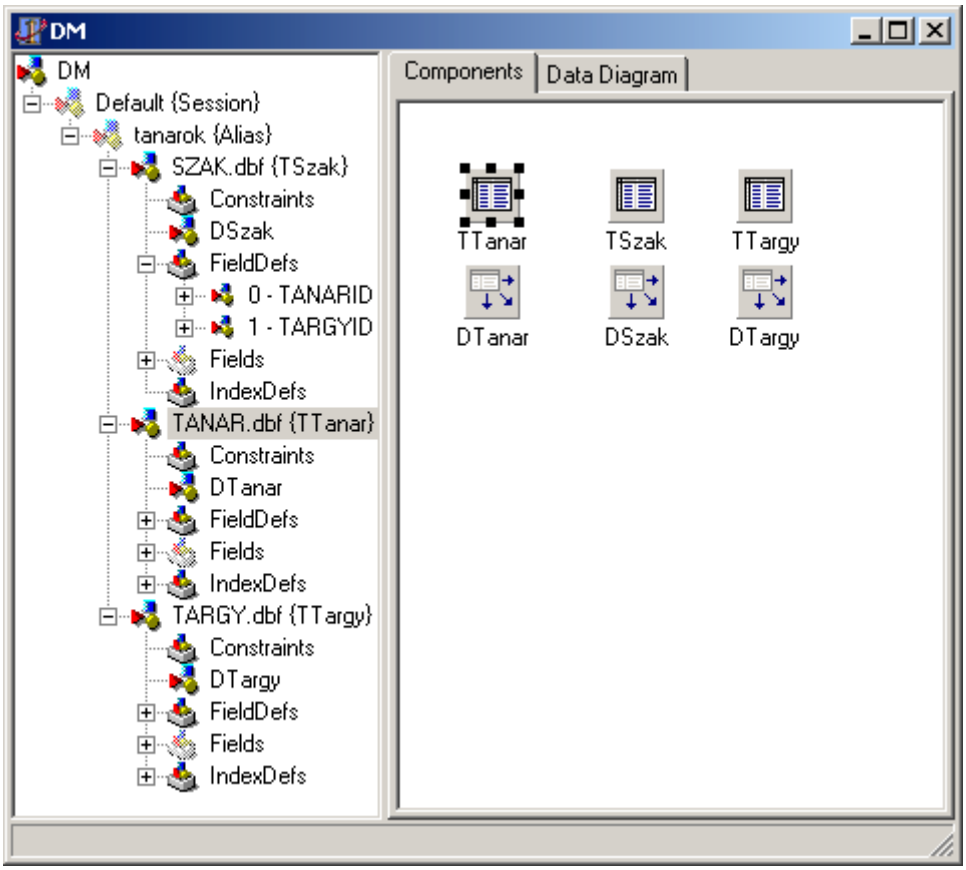


Vigyük fel a további két táblánkat is (TARGY, SZAK), ne feledkezzünk el az indexelésekről: TARGY.ID, SZAK.TANARID, valamint SZAK:TARGYID. Mentések után töltsünk fel egy-egy adatot (a korábbiaknak megfelelően) minden táblába.

Meg kell még adnunk az aliast, mellyel elérhetjük a létrehozott tábláinkat:

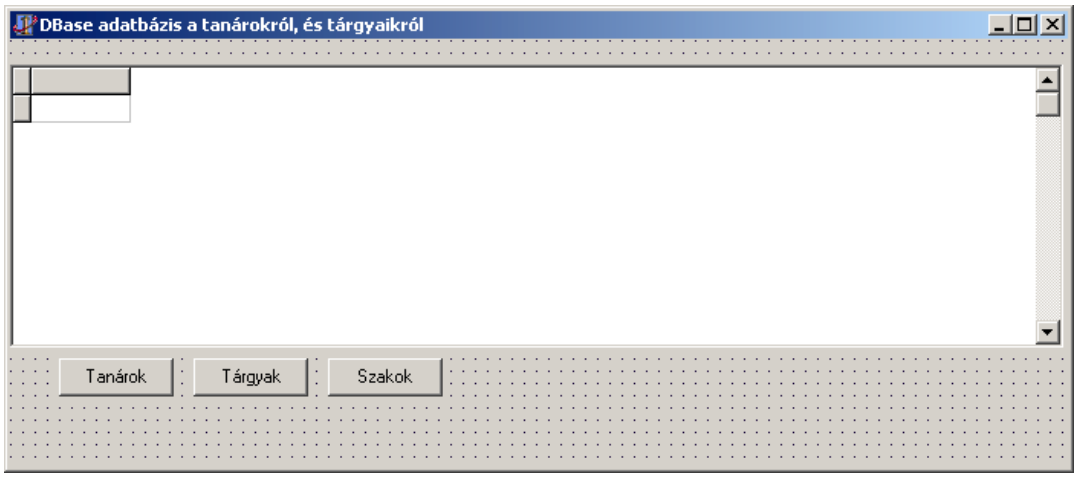
5. Menjünk a Tools menü Alias Manager almenübe, majd a NEW gombbal definiáljunk új aliast. „tanarok" néven, a DBASE könyvtárat kiválasztva. Hagyjuk jóvá a Keep New gombbal, majd az OK megnyomásával vegyük tudomásul az új alias bekerülését az Idapi.cfg-be. Be is zárhatjuk a Database Desktop-ot, és visszakerülünk a Delphi felületére.

6. A most következő elemünk egy adatmodul lesz, melyet azért veszünk fel, hogy a kezelőfelületen a háttérben lévő modulra tudjunk hivatkozni, elkerülve ezáltal a kezelőfelület kusza rendezetlenségét. Válaszuk a File menü New lehetőségét, és a megjelenő ablakból a második sorban található Data Modul-t. Állítsuk be a Name tulajdonságot: DM. Erre a modulra helyezzük el a tábláinkat, és minden táblához egy-egy DataSource-t. A táblák nevei: Ttabel (tanár), Tszak (szak), Ttargy (tárgy). Rendeljük hozzájuk az adatbázist (aliasunkat), és állítsuk be a hivatkozásaikat a táblákra. Tegyük végül aktívvá őket.
7. Vegyük fel a táblák alá a DataSource elemeket Dtanar, Dszak, Dtargy és irányítsuk őket a tábláinkra.
8. Ha elkészültünk, kérjünk mentést a Save All-al, ekkor egy Unit mentését fogja felkínálni a rendszer (ez a most létrehozott adatmodulunk) mentsük UDM néven a könyvtárunkba.

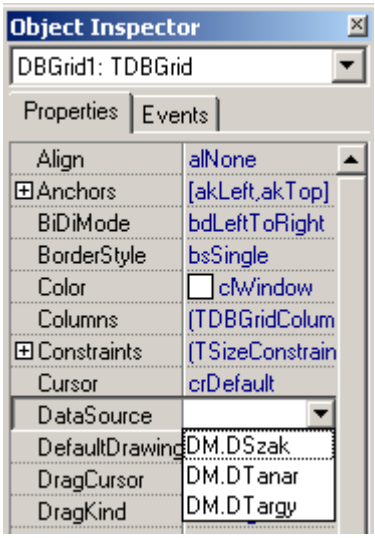


Ahhoz, hogy modulunkat el tudjuk érni, hivatkoznunk kell majd rá a kódunkból, amit a USES deklarációban tehetünk meg.

9. Lépjünk át a főablakunkba, és helyezzük el a komponenseinket a formon. Már nem lesz szükségünk a táblákra, adatforrásokra, mert azok ott vannak a háttérben. Tegyünk fel egy DBGrid-et a form tetejére, lent pedig helyezzük el a korábban már megismert három gombot a korábbi beállításokkal.



10. Mint látható, a Grid-hez nem fogunk tudni dataSource-ot beállítani (próbáljuk ki, ne higgyünk a jegyzetnek!), ugyanis a jelenlegi unitunkban nincsen ilyen. Kattintsunk a háttérben lévő kód ablakába, és a görgetősávval keressük meg a USES listát. Egészítsük azt ki az adatmodulunkat tároló unit nevével (UDM), és kérjünk mentést (Save All). Visszatérve a főablakhoz, már be fogjuk tudni állítani az adatforrást. Legyen ez alapértelmezetten a tanáraink adatait tartalmazó DataSource. Ha mindent megfelelően hajtottunk eddig végre, akkor a listában ott fog szerepelni a három háttérben lévő elemünk neve:



11. Ha hozzárendeljük a DM.Dtanar adatforrást a Grid-hez, akkor azonnal meg fog jelenni a korábban felvitt elemünk a Gridben. Írjuk még meg a gombok kódjait is, a korábbi módszerrel, vagyis futásidőben a

```
DBGrid1.DataSource:=DM.Dtargy;
DBGrid1.DataSource:=DM.Dszak;
```

Mentsünk, futtassunk, és győződjünk meg róla, hogy minden rendben van-e. Ha igen, akkor elkészült a korábbi adatbázisunk alapvető kezelő felülete. Itt ugyanazt mondhatjuk el, mint korábban, vagyis ez egy alapja lehet egy komolyabb rendszernek, de természetesen még kiegészítésekre, hibakezelésekre szorul.

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Az Interbase szerverre történő bejelentkezéshez mit nem kell megadnunk?
   a., Felhasználói név, jelszó
   b., Távoli szerver elérési útvonala
   c., Távoli, vagy helyi szervert használunk
2. Melyik elem nem része az Interbase komponenspalettának?
   a., IBTransaction
   b., IBImage
   c., IBSQL
3. Hány elemre van minimum szükségünk, ha egy Interbase tábla valamennyi adatát szeretnénk megjeleníteni?
   a., 3
   b., 5
   c., 7
4. Mi a kiterjesztése egy Dbase állománynak?
   a., MDB
   b., DB
   c., DBF

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. Az Interbase adatbázis egyetlen állományban tárolódik.
   igaz
   hamis
2. A Dbase adatbázis egyetlen állományban tárolódik.
   igaz
   hamis
3. Az Interbase adatbázis megjelenítésére külön IBGrid komponens szolgál.
   igaz
   hamis
4. Interbase alapú adatbázisnál nem használhatunk adatmodult.
   igaz
   hamis
5. A Database Desktop csak a Delphiből indítható el.
   igaz
   hamis

Adatbáziskezelés IV.

Eddigi adatbázisos feladatainkból az alapokkal ismerkedhettünk meg. Jelen fejezetben egy kicsit tovább szeretnénk finomítani az ismereteket.

Lekérdezések (Query)

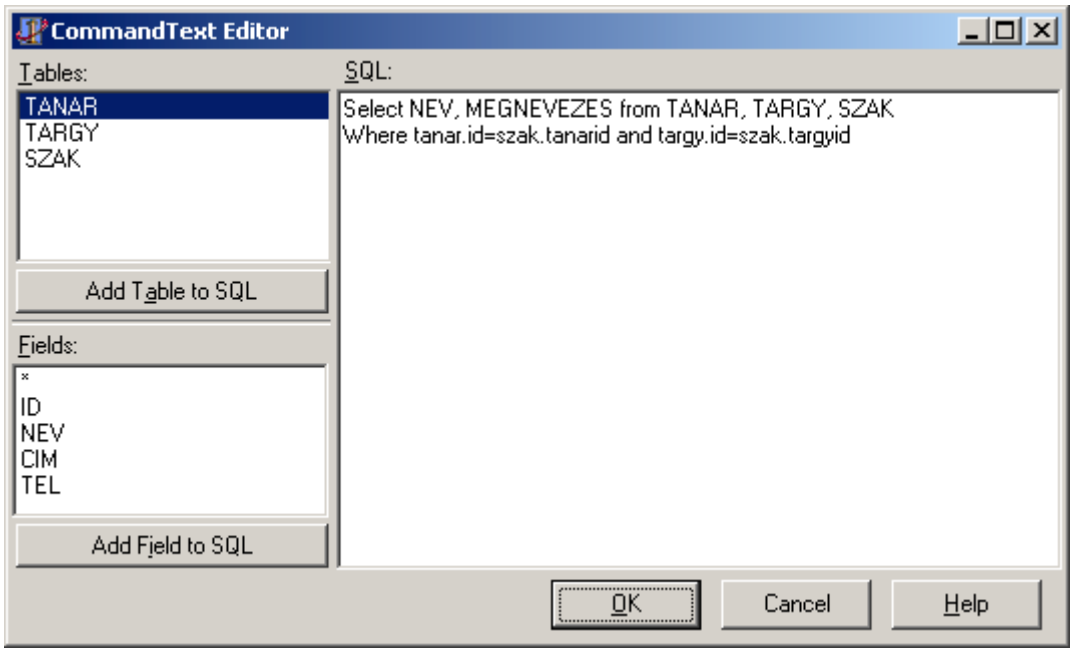
Az előző fejezetekben utaltunk már arra, hogy nem elegendő egy adatbázisban az adatok egyszerű tárolása, szükséges azok elérésének valamely szempont alapján történő szűrésének a lehetőségét is biztosítanunk. Ehhez a korábbiakban már megismert SQL nyelvet hívhatjuk segítségül, bármely felületen dolgoztunk is az adatbázisban. Első feladatunkban a korábban már létrehozott TANTARGY (InterBase alapú) adatbázisunkat fogjuk felhasználni.
Talán emlékszünk még, hogy ebben az adatbázisban nem volt szükségünk az Alias létrehozására, hisz a projectünk egyetlen könyvtárban volt, ahol maga az adatbázis is elhelyezkedett. Keressük meg tehát a könyvtárunkat az intézőben (TANTARGY), majd nyissuk meg az ott található Pfo.dpr állományt, mely a delphi project állománya. Ha elindult a Delphi, akkor csinosítsuk egy kicsit az alkalmazásunkat.

1. Helyezzünk el egy MainMenu komponenst (Standard) a formunkon, hogy lehetőséget biztosítsunk a felhasználónak a leendő lekérdezéseink elérésére.
2. A ManiMenu1-re duplát kattintva hozzuk létre a következő menü szerkezetet:

FILE
    Kilépés
ŰRLAPOK
    Tanárok
    Tantárgyak
    Szakok
LEKÉRDEZÉSEK
    Tanárok tárgyai
    Tárgyak tanárai
    Paraméterezett

Ha elkészültünk a menüvel, zárjuk be a párbeszédpanelt, majd lépjünk az új menünkbe, és írjuk meg a kódokat.

3. A kilépés menü hívja meg a close metódussal a bezárást.
4. Az űrlapok lehetőségei ugyanazokat a kódokat fogják tartalmazni, mint a lentebb látható gombok, így hozzájuk tudjuk rendelni a megfelelő eseményeket. Csak ismétlésképpen: ha már megírtunk egy eljárást, azt bármikor meg tudjuk hívni a kódból, tehát nincs szükségünk annak újbóli megírására. Esetünkben azonban (mivel maga a kód szinte rövidebb, mint az esemény meghívását kezdeményező utasítás) akár meg is írhatjuk újra az eljárásokat. Ezzel a menü első része elkészült, mentsünk, majd próbáljuk ki azt.
5. Visszatérve a tervezéshez, keressük meg az Interbase komponenspalettán található IBQuery-t, és helyezzük el a formunkon a DBGrid felett a tábláink mellé. A Database tulajdonságnál állítsuk be az egyetlen lehetséges IBDataBase1-et, majd keressük meg az SQL (Tstrings) tulajdonságot, és kattintsunk a megjelenő három pontra (…). Írjuk be a párbeszédablakba a következő SQL utasítást:



Az utasítás részletes magyarázatától eltekintenénk, ezt korábban már egy másik tárgy keretén belül megtettük. Az utasításunk eredménye tulajdonképpen nem más, mint a szak táblában található adatoknak a megjelenítése „értelmezhető” formában.
Mint fentebb is látható, az ablakban az Interbase táblák, és a hozzájuk tartozó valamennyi mező megjelenik, ezzel leegyszerűsítve a felesleges SQL kódolást a gép. Használjuk a mezők felviteléhez a segítséget.

6. Tegyük aktívvá a lekérdezésünket (Active: True), majd a menünk első lekérdezésének a kódjához írjuk be a Query-t a DBGrid sorforrásának.

```
procedure TfrmFo.Tanroktrgyai1Click(Sender: TObject);
begin
DataSource1.DataSet:=IBQuery1;
end;
```

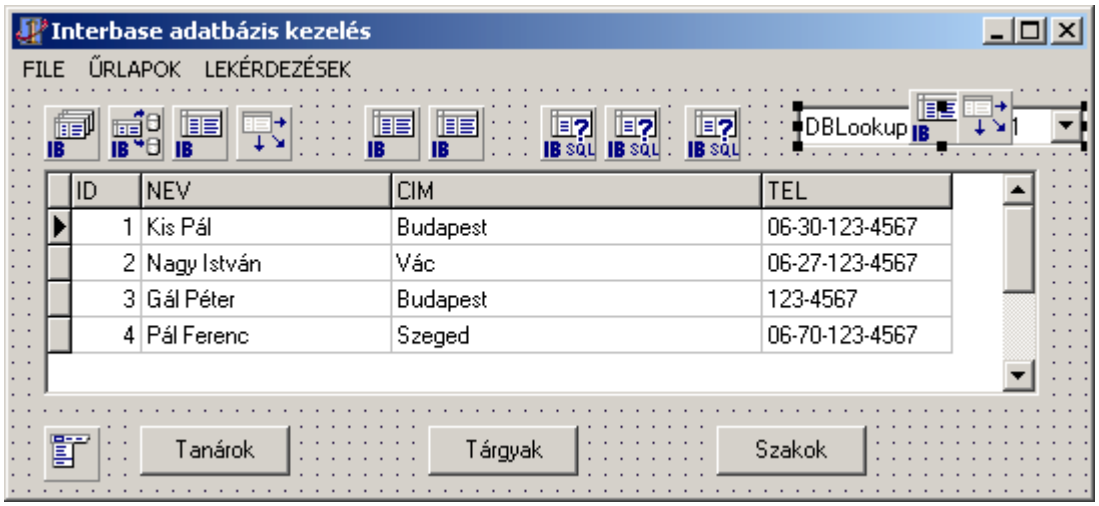
Mentés után futtassunk, majd győződjünk meg róla, hogy minden rendben megjelent.

7. Természetesen érdemes néhány további adatot felvinni az adatbázisba, hisz jelenleg egyetlen tanár adatai jelennek meg, akit még az Interaktív SQL segítségével vittünk fel az Interbase Manager felületén. Ezt oldjuk meg a felületünkön keresztül.
8. Készítsük most el önállóan a tárgyak tanáraihoz tartozó Query-t, majd rendeljük azt hozzá a menüponthoz.
9. Természetesen az SQL szabályai érvényesülnek itt is, így mindaz, amit korábban megtanultunk ezzel kapcsolatban itt is hasznosítható. Rendezzük például a második query megjelenítendő adatait a tárgyak szerint növekvő sorrendbe (egészítsük ki az SQL utasításunkat a GROUP BY 1 utasítással). Ne felejtsük el aktívra állítani ezután a lekérdezést (minden módosításkor automatikusan bontja a kapcsolatot az adatbázis, és az aktív elem között a delphi), majd mentés után futtassunk.
   Nézzünk most egy összetettebb feladatot:

Paraméterezett lekérdezés

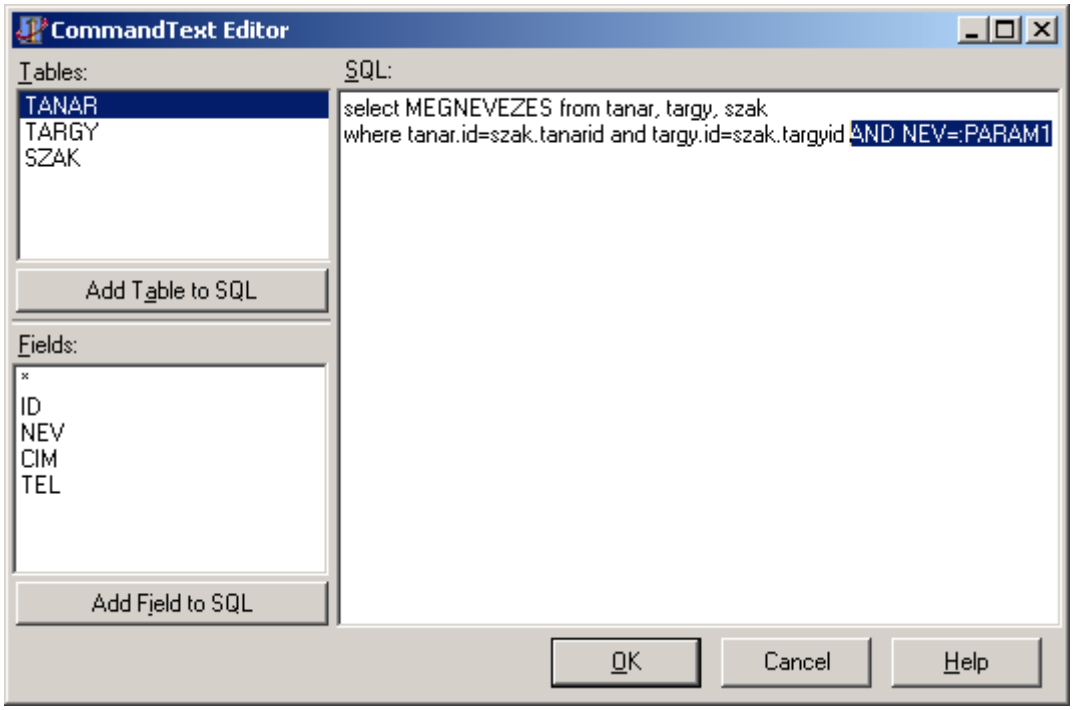
Ahhoz, hogy egy paraméterezett lekérdezést tudjunk futtatni, el kell döntenünk, miképpen kérjük be a használni kívánt paramétert. Ez lehetséges egyrészről egy beviteli mező felkínálásával, ahol a felhasználó megadhatja a kívánt paramétert, egy InputBox-al, egy kombinált listával, illetve egyéb vezérlők segítségével. Tételezzük fel, hogy a tantárgyakat szeretnénk megjeleníteni, de csak annál a tanárnál, akit a felhasználó egy paraméter segítségével megad.
A beviteli mezőnek megvan az a hátránya, hogy a felhasználó (akarva, akaratlanul) elkövethet gépelési hibákat, így a bevitt paraméter nem lesz megfelelő. Miután adatbázisunk tartalmazza a tanárok adatait, így nevüket egyszerűbb megjeleníteni egy kombinált listában, csökkentve ezzel a hibák lehetőségét. Természetesen ez jelentősen meg fogja növelni a tennivalónkat, ráadásul viszonylag nagy mennyiségű további elem elhelyezését is megköveteli. Nézzük a teendőket:

10. Helyezzünk el a DBGrid jobb felső részében egy DBLookupComboBox (Data Controls), valamint egy DataSource (Data Access) elemet.
11. A DataSource2 DataSet tulajdonságát állítsuk rá a tanár táblánkra, melyet az IBTable1 tartalmaz.
12. A DBLookupComboBox1 ListSource tulajdonsága most legyen a DataSource2, és a ListField tulajdonság pedig legyen a NEV, a KeyField pedig az ID. Jelenleg valahogy így állunk:



Láthatóan törekedtünk az áttekinthető felületre, de így is eléggé érzékelhető a hiányosság: már el kellett volna gondolkodnunk egy DataModul elkészítésén. Mivel azonban itt már nincs sok teendőnk hátra, így eltekintünk ennek létrehozásától, de nyilvánvalóan érdemes ezt elkészíteni, amennyiben előre látható hogy sok elem, és adat lesz a megjelenítendő felületen.
Elkészült tehát a forrás, készítsük most el a hozzá tartozó lekérdezést, és a paraméteres szűrést.

13. Helyezzük el az utolsó SQL elemünket, rendeljük hozzá az adatbázist, és hívjuk elő az SQL szerkesztőt. A korábban már megírt SQL utasításunk annyiban módosul, hogy kiegészítjük egy Param1 sorral a képen is látható módon:



Vegyük észre, hogy a NEV mezőnél a paraméterre történő hivatkozás „=:”. Ezzel az utasításunkkal tulajdonképpen már létre is hoztuk a paraméterünket (ne higgyünk a jegyzetnek, nézzük meg bezárva az ablakot az SQL feletti Params tulajdonságot), de természetesen még nem rendelkezik értékkel, erről majd még gondoskodnunk kell a megfelelő helyen, illetve időben, némi hibakezeléssel együtt. Most mentsünk, futtassunk, és győződjünk meg róla, hogy a listában megjelennek az elemek.

14. Válasszuk ki a DBLookupComboBox elemünket, és keressük meg az OnCloseUp eseményt, melynek bekövetkeztekor szükségünk lesz a lekérdezésünk újrafuttatására, amit csak annak kikapcsolása, majd újranyitása tesz lehetővé, valamint a paraméterünket is itt fogjuk beállítani. Írjuk tehát be a kódot:

```
procedure TfrmFo.DBLookupComboBox1CloseUp(Sender: TObject);
begin
IBQuery3.Params[0].text:=dbLookupcomboBox1.Text;
IBquery3.close;
IBquery3.open;
end;
```

15. Lépjünk most a lekérdezés menünkre, és válasszuk ki a harmadik (paraméteres) menüpontot. Mi lesz a teendőnk? Tulajdonképpen semmi más, mint hozzárendelni a Gridhez a Query-t, feltéve, hogy a kombinált lista már tartalmaz kiválasztott elemet. Maga a szűrés a listában történő módosítás után azonnal megtörténik, így a hibakezelés itt az elején válik csak szükségessé. De mint látni fogjuk ez sem okoz hibát, csupán nem jelennek meg adatok, amennyiben a felhasználó nem válasz ki tanárt a listából. Kódunk a következő:

```
procedure TfrmFo.Paramteres1Click(Sender: TObject);
begin
DataSource1.DataSet:=IBQuery3;
end;
```

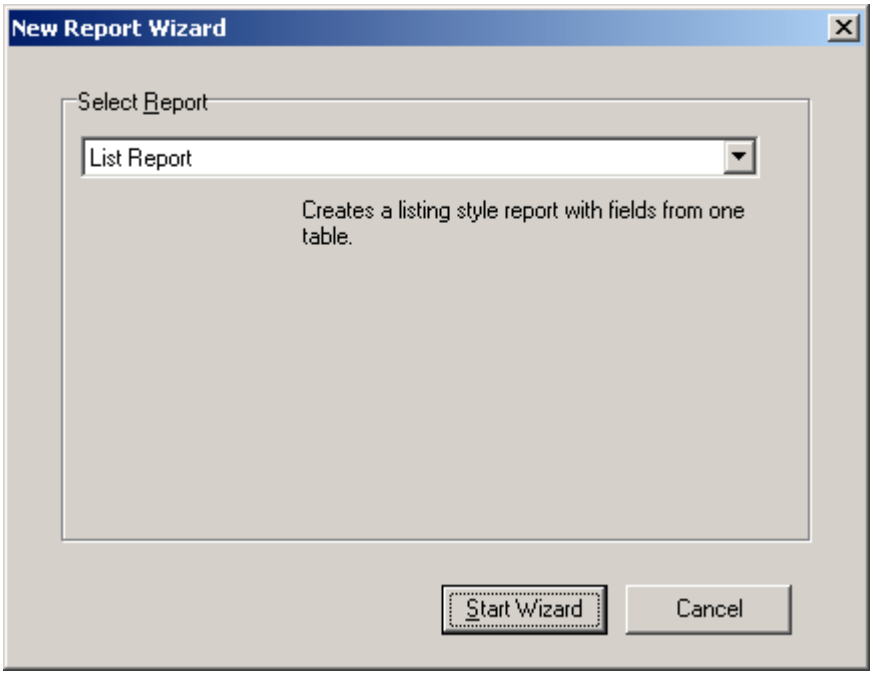
És már el is készültünk paraméterezett lekérdezésünkkel, mely a felhasználó által választott tanárhoz tartozó tárgyakat fogja kilistázni a képernyőre.
Ha elkészültünk teszteljük az adatbázisunkat, majd mentsünk, és zárjuk be azt.

Jelentések (Quick Report)

Maradt még egy fontos feladatunk: lehetőséget kell biztosítanunk a felhasználóknak arra, hogy az adatbázis adatait nyomtatott formában is képesek legyenek megjeleníteni.
Jelentések létrehozásához a Delphiben ismét segítségül hívhatjuk a varázslót.
A jelentések esetében annyit kell tudnunk, hogy önálló Unitként jönnek létre, így amennyiben nincsen project a futtatásuk nem lehetséges. Ha viszont van projectünk (alapértelmezetten indításkor létrejön), akkor problémát jelenthet, hogy az alapértelmezett Unit neve is UNIT1, és a varázsló is ezen a néven igyekszik létrehozni a jelentéshez kapcsolódó Unit-ot. Ezt elkerülhetjük, ha az indításkor rögtön figyelünk arra, hogy a megfelelő mentéseket végrehajtsuk. Nézzük, tehát a varázsló használata esetén mit kell tennünk:

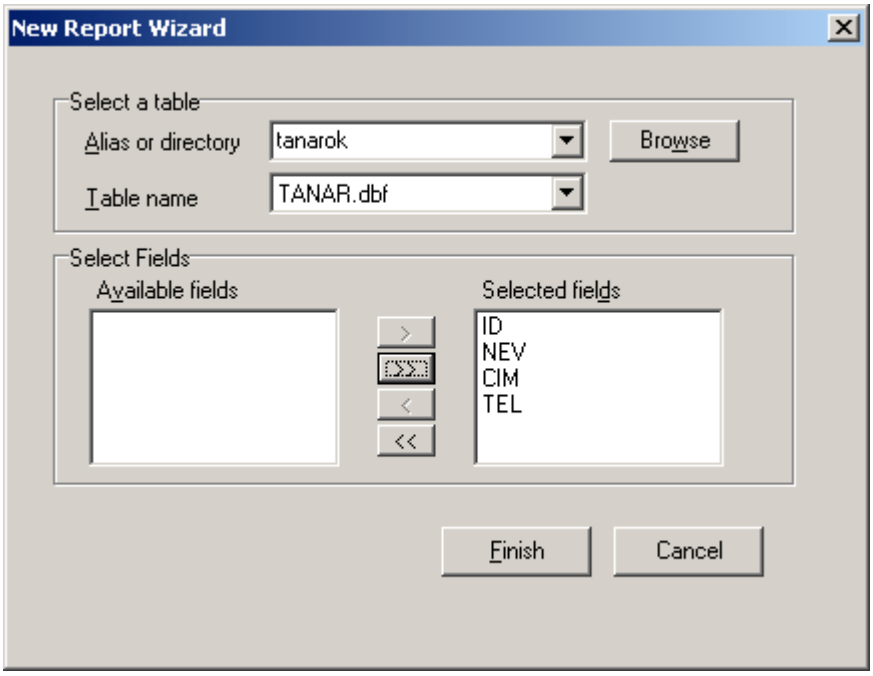
Jelentés készítése varázslóval (JELENTPRO)

1. Hozzuk létre a könyvtárat, indítsuk el a Delphit, a formunkat nevezzük frmProba-nak, majd kérjünk egy mentést: Uproba, Pproba a létrehozott JELENTPRO könyvtárba.
2. Menjünk a File menü New almeüjéhez, a megjelenő ablakban válasszuk a Business lapot, és ott indítsuk el a QuickReport Wizard-ot. A következő képernyőt kellene látnunk:



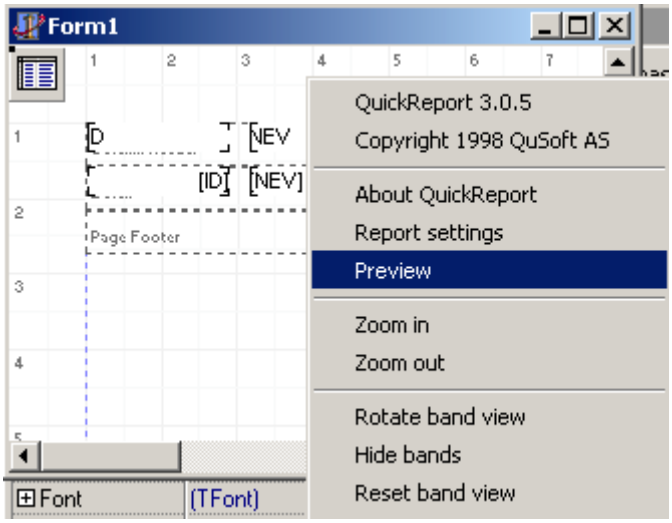
A varázslóban a lenyíló lista csak a List Reportot tartalmazza, indítsuk a varázslót.

3. A megjelenő ablakban válasszuk ki az Aliast (ez legyen a tanarok), válasszuk ki a táblát (TANAR.DB), és a megjelenő mezőket dobjuk át a szokott módon a jobb oldali ablakba:



Mint látható, csupán a Finish maradt, így fejezzük be a varázslót.

4. Rögtön meg kellett jelenjen egy Form1 néven futó új ablak, és természetesen a hozzátartozó Unit is létrejött (Unit1 néven). A jelentés megtekintéséhez elegendő jobb egérgombot nyomnunk a megjelent új form felső jelentésterületén, és a gyorsmenüből kiválasztani a Prewiev lehetőséget:



Ennek hatására a jelentésünk nyomtatási képe válik láthatóvá. Előnye, hogy nem szükséges az alkalmazásunkat futtatnunk ahhoz, hogy a változtatásaink eredményét megtekintsük.

A jelentésünk tehát elkészült, csupán az lenne a feladatunk, hogy megjelenítsük azt valamilyen módon a felhasználók számára is. Ehhez a következőt kell tennünk:

5. Az előttünk lévő Form1 tulajdonságait válasszuk ki az objektum-kezelő lenyíló listájából, majd a Name tulajdonságot állítsuk át frmJelent-re. Kérjük a Save All-t, majd a megjelenő párbeszédpanelben mentsük el a jelentésünk forrását Ujelent néven a könyvtárunkba.
6. Hívjuk elő a főformunkat (frmProba), és helyezzünk el rajta egy Button-t. Name: btnJelent, Caption: Jelentés indítása.
7. Kattintsunk duplán a létrehozott gombunkra, és írjuk be a következő kódot:

```
procedure TfrmProba.btnJelentClick(Sender: TObject);
begin
frmJelent.QuickRep1.preview;
end;
```

A kódban a jelentésünk formján lévő alapértelmezetten QuickRep1-nek nevezett jelentésünk nyomtatási kép nézetben történő megjelenítését kezdeményeztük. A kód írása közben talán tapasztalhattuk, hogy nem segített a delphi, ennek magyarázata egyszerű: egy másik Unitban lévő objektummal kívántunk műveletet végrehajtani. Természetesen ez számára még nem látható. A problémát orvosolhattuk volna, ha a kódunk írása előtt felvesszük a használni kívánt Unit-ot (esetünkben Ujelent) a USES listába. De mi bíztunk abban, hogy a fent látható utasítást sikerülni fog begépelni gond nélkül. Nézzük is meg:

8. Mentsünk, és próbáljunk futtatni. A Delphi jelezni fogja, hogy az általunk használni kívánt Unit nem szerepel a listában, természetesen igennel válaszoljunk. Ezt kövesse egy újabb mentés (hisz felülírtuk a kódot, még akkor is, ha ezt a Delphi tette helyettünk).
9. A futtatás során a jelentésünk meg fog jelenni a felhasználónak a megfelelő adatokkal, és mint látható, a nyomtatáshoz szükséges alapvető elemek is elérhetővé válnak számára.

Az egyetlen apró figyelmeztetés: ahogy korábban (még az Access adatbázis-kezelőnél) a jelentésekkel kapcsolatban megjegyeztük, vigyázzunk, hisz a jelentés közvetlenül nyomtatóra is küldhető. A példafeladat jelentése ugyan nem fog jelentős mennyiségű tintát, és lapot fogyasztani (nem véletlenül választottuk ezt gyakorláshoz), de egy komolyabb adatbázis esetében, hálózati nyomtatóra nyomtatva elég komoly szemrehányásokat kaphatunk.

Első jelentésünk tehát elkészült, mint láttuk egy táblából vettük az adatokat, és egy önálló felületen jelenítettük meg azokat. Tekintsük át, tehát miből épül fel a jelentés, és milyen eszközöket biztosít számunkra az elkészítéséhez a Delphi.

A Qreport komponenspaletta:

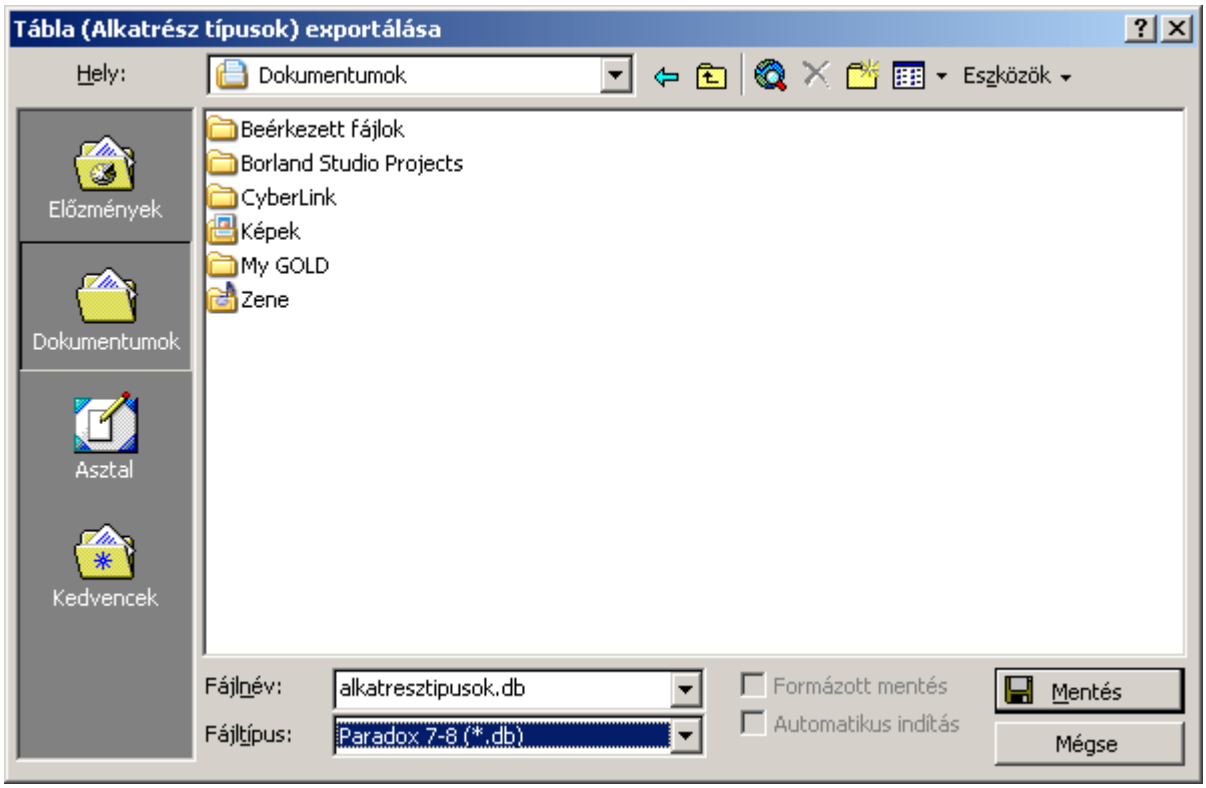
Bár talán azt gondolhatnánk, jelentéktelen objektumról van szó, ha megnézzük mégis látható, hogy szinte a legtöbb eleme ennek a palettának van.
QuickRep: maga a jelentés objektum, mely tartalmazza az elemeket.
QRSubDetail: Fő, és segédjelentéshez.
QRStringsBand: Szövegcsoport, információk megjelenítésére.
QRBand: A jelentések csoportszintjei.
QRChildBand: Alcímek a jelentésben.
QRGroup: Csoportosítások a jelentésekben.
QRLabel: Címkék a jelentésen.
QRDBText: Adatbázis mezők a jelentésben.
QRExpr: Kifejezések, számított értékek a jelentésben.
QRSysData: Rendszeradatok megjelenítése (dátum, oldalszám, stb)
QRMemo: Feljegyzés a jelentésen.
QRExprMemo: Összetett feljegyzések.
QRRichText: Szöveges információk megjelenítése.
QRDBRichText: és ugyanez egy adatbázisból.
QRShape: Egyszerű primitívek (kör, vonal, stb) létrehozásához.
QRImage: Képek a jelentésen,
QRDBImage: esetleg adatbázisból.
QRCompositeReport: Kapcsolódási lehetőség más jelentésekhez.
QRPreview: Valós idejű nyomtatási kép megjelenítés.
QRTextFilter: A jelentés ASCII formában történő elmentésére ad lehetőséget, felkínálva a mentés párbeszédablakát.
QRCSVFilter: A jelentés CSV (tagolt táblázatos) formában történő elmentésére ad lehetőséget.
QRHTMLFilter: A jelentés HTML oldalként történő mentésére ad lehetőséget.
QRChart: Diagramot készíthetünk jelentésünkről.

A fenti komponensek az eddig megismert módon használhatóak, vagyis a megfelelő elemeket a kívánt helyre elhelyezve, tulajdonságaikat beállítva készíthetjük el a kívánt dokumentumot, jelentést.
Következő összetettebb példánkban ezt mutatjuk be.

Feladat: Számlajelentés (SZAMLA)

Ismét utalnánk a korábban már elkészített Access számítógépbolt adatbázisra. Mint emlékszünk, a számla jelentés elkészítése nem volt egyszerű feladat.
Most, hogy ismét egy kis kiegészítő ismeretet is szerezhessünk, készítsük el a számítógépbolt adatbázisunk Access formátumból történő átalakítását Paradox formátumba. Hozzunk először létre egy könyvtárat a Delphi könyvtárunkon belül, legyen a neve SZAMLA.

1. Indítsuk el az Access adatbázis kezelőt, és tallózzuk ki a számítógépbolt adatbázisunkat. Ha megjelent az adatbázis ablak, lépjünk át a táblákhoz, és válasszuk ki az alkatrész típusok táblát. Kattintsunk a táblánkon jobb egér gombbal, a megjelenő menüből pedig válasszuk ki az Exportálás lehetőséget.
2. A párbeszédablakban adjuk a táblának ékezetek nélkül egy szóban az alkatresztipusok nevet, majd az állomány típusánál (Fájl típus) adjuk meg a Paradox 7-8 típust. Az útvonal legyen az előzőleg létrehozott SZAMLA könyvtárunk, ide fogjuk exportálni a kiválasztott táblánkat.



Ha végeztünk, kérjük a mentést, majd győződjünk meg róla, hogy SZAMLA könyvtárunkban megjelent az adatbázis állomány: alkatresztipusok.DB

3. Exportáljuk még a számlánkhoz szükséges táblákat, vagyis a következő adatbázis állományokat kellene létrehoznunk:

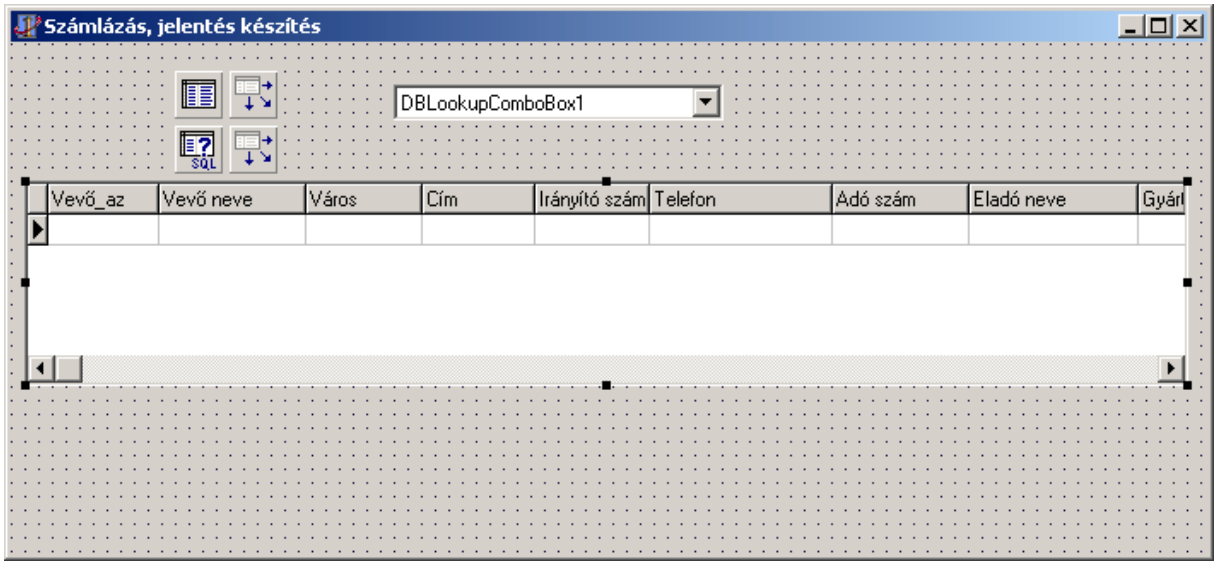
ALKATRESZTIPUSOK.DB
ELADOK.DB
GYARTOK.DB
RAKTARKESZLET.DB
SZALLITOK.DB
SZAMLAK.DB
SZAMLAKRESZLETEI.DB
VEVOK.DB

Tulajdonképpen ettől kezdve már minden a megszokott módon történhet, vagyis kezdhetjük a tábláink megjelenítéséhez szükséges felület kialakítását.

4. Indítsuk el a Delphit, és alapozzuk meg a számlázást. A form Name: frmSzamla, a Unit neve: Uszamla, a project Pszamla.
5. Hozzuk létre a „szamla" aliast, hogy egyszerűsítsük a dolgunkat. Indítsuk el a Tools menü Database Desktop lehetőségét. Itt válasszuk a Tools Alias manager-t. Kattintsunk a New gombra, fent adjunk Alias nevet, tallózzuk ki a SZAMLA könyvtárat, majd kattintsunk a Keep New gombra. Ha ez megvolt, akkor OK, ekkor kérdezni fogja tőlünk a gép, hogy felvesszük-e a listába az új aliast, természetesen, hisz ezért jöttünk ide. Bezárhatjuk a Database Desktopot, és visszatérhetünk a Delphihez.
6. Gyakorlás képpen a korábbi példából már megismert módon fogjuk a számlázni kívánt számlát kiválasztani, vagyis paraméterezett lekérdezést hozunk létre. Helyezzünk el egy táblát (Data Access Table) a formon, Database Name: szamla, TableName: szamlak.db, és végül az Active: True.
7. Jöjjön egy DataSource (Data Access), DataSet tulajdonsága legyen a Table1.
8. Jöhet egy DBLookupComboBox (Data Controls), ListSource: DataSource1, ListField: Számla_az, és KeyField: Számla_az. Ha mentünk, és futtatunk, meg kell hogy jelenjenek a számlák azonosítói.
9. Jöjjön a lényeg: szeretnénk összegyűjteni a számlához szükséges valamennyi adatot egy lekérdezés segítségével, melyet majd szűrni (paraméterezni) szeretnénk a felhasználó által kiválasztott számlára.
   Helyezzünk el egy Query (Data Access) komponenst a formon, kattintsunk az SQL TStrings tulajdonságán, és írjuk be a következő kódot (ha nagyon nem érthető ami itt következik, javasolt az SQL jegyzet ismételt tanulmányozása):

```
SELECT DISTINCT D.Vevő_az, D."Vevő neve", D.Város, D.Cím, D."Irányító szám", D.Telefon, D."Adó szám", D1."Eladó neve",
D2."Gyártó neve", D3.Ár, D3.Leírás, D5.Számla_az, D5.Sztornózott, D5.Dátum, D5.Átutalás, D6."Gyártási szám", D6.Darab,
D7.Megnevezés
FROM ":szamla:vevok.DB" D, ":szamla:eladok.DB" D1, ":szamla:gyartok.DB" D2, ":szamla:raktarkeszlet.DB" D3,
":szamla:szallitok.DB" D4, ":szamla:szamlak.DB" D5, ":szamla:szamlakreszletei.DB" D6, ":szamla:alkatresztipusok.DB" D7
WHERE
(D3.Gy_az = D2.Gy_az)
 AND (D4.Sz_az = D3.Sz_az)
 AND (D5.Vevő_az = D.Vevő_az)
 AND (D5.Eladó_az = D1.Eladó_az)
 AND (D5.Sztornózott = 0.0)
 AND (D6.Számla_az = D5.Számla_az)
 AND (D6.Rk_az = D3.Rk_az)
 AND (D7.At_az = D3.At_az)
 AND (D5.Számla_az=:alma)
ORDER BY D.Vevő_az, D."Vevő neve", D.Város, D.Cím, D."Irányító szám", D.Telefon, D."Adó szám", D1."Eladó neve",
D2."Gyártó neve", D3.Ár, D3.Leírás, D5.Számla_az, D5.Sztornózott, D5.Dátum, D5.Átutalás, D6."Gyártási szám", D6.Darab,
D7.Megnevezés
```

   Annyit kell még tennünk, hogy a Query-ben létrehozott alma paraméternek a típusát megadjuk, legyen ftInteger. Állítsuk most true-ra a Query Active tulajdonságát.
10. Helyezzünk el egy újabb DataSource elemet a formon, és állítsuk rá a Query-re.
11. Ellenőrzésképpen jöjjön még egy Grid, ami most vegye az adatokat a DataSource2-ből. Ezt csupán ellenőrzésképpen helyezzük el, a gridben meg kell, hogy jelenjenek a mezőnevek tervezés alatt! Ahogy ezt az ábrán is láthatjuk:



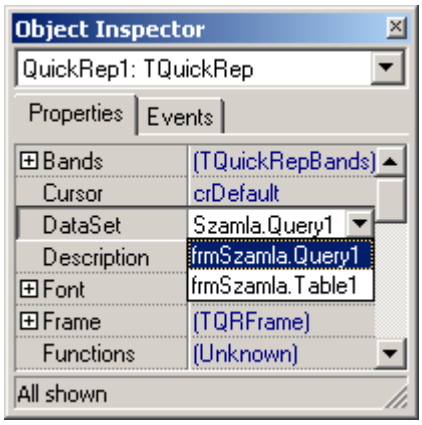
Természetesen futásidőben még nem azt látjuk amit szeretnénk, mert a paraméterezésünket nem fejeztük be.

12. Szűrjük most akkor meg az adatokat, válasszuk ki a DBLookupComboBox OnCloseUp eseményét, és írjuk be a következő kódot:

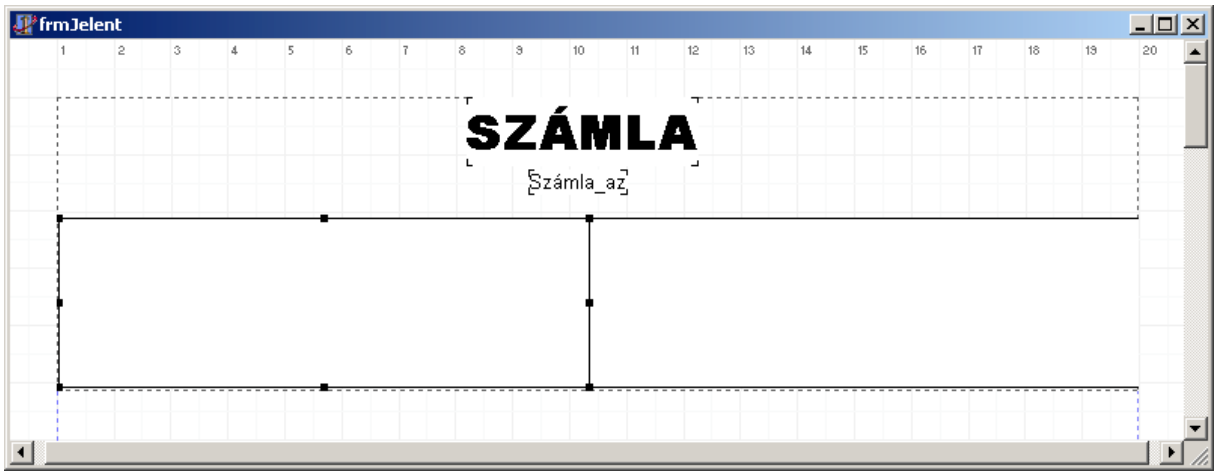
```
procedure TfrmSzamla.DBLookupComboBox1CloseUp(Sender: TObject)
begin
 Query1.params[0].Text:=dblookupcombobox1.text;
 query1.Close;
 query1.open;
end;
```

Ha mentünk, és futtatunk, akkor a paraméternek megfelelő számlát fogjuk látni a Gridben, és ez minden újabb választás esetén megtörténik. Vagyis szűrtük az adatokat, jöhet a szűrt adatokkal történő jelentés megjelenítése:

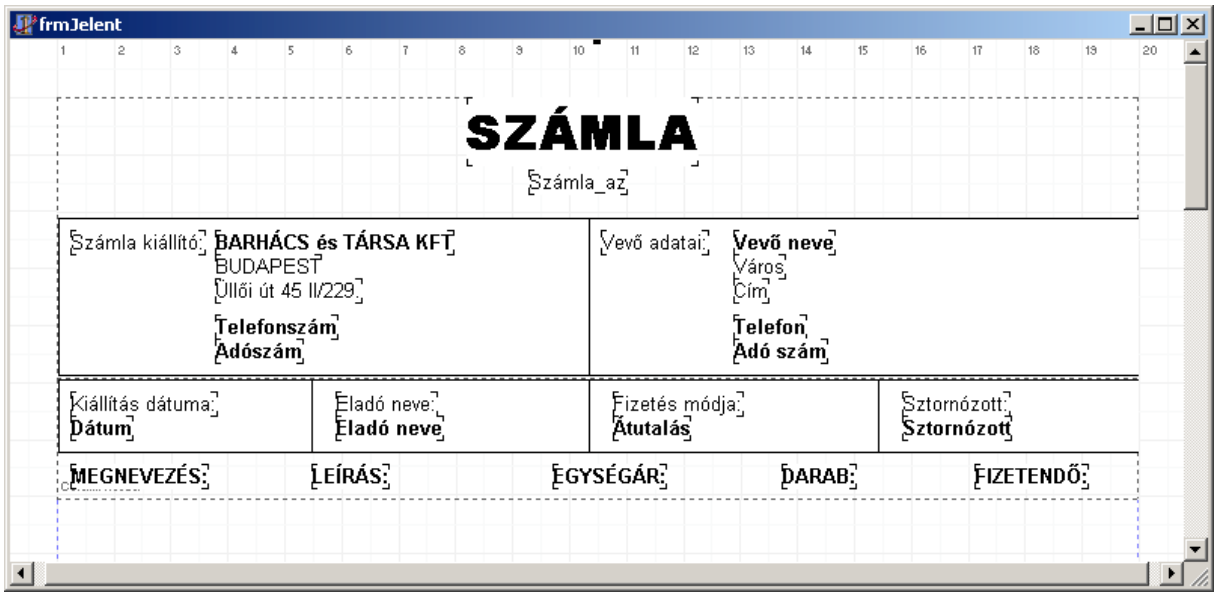
13. Mivel most mindent saját kezűleg készítünk el, így apró lépésekből fog felépülni a jelentésünk. Válasszuk először a Delphi File menüjében a New Form-ot, ezen helyezzük majd el a Quick Reportot. Erre azért van szükségünk, mert a jelentés objektum felülete látható, így vizuálisan jelenik meg futásidőben is, ami zavaró lenne a főablakban. Az új formunknak adjuk az frmJelent nevet.
14. Kérjünk mentést (Save All), és a megjelenő párbeszédablakban mentsük el az újonnan létrehozott ablakunk kódját Ujelent néven.
15. Lépjünk át a Qreport palettára, és az első elemet (QuickRep) helyezzük el a form bal felső sarkába kattintva. Ha megpróbáljuk a DataSet tulajdonságot beállítani, akkor látható, hogy a lista üres, tehát nem talál a rendszer elérhető forrást. Ha begépelnénk, kézzel a hivatkozást, az szintén hibaüzenettel járna. A szokásos probléma: nem hivatkoztunk a használni kívánt Unit-ra, tehát keressük meg az Ujelent kódját, és a USES listában hivatkozzunk az Uszamla Unit-ra. Ezt követően (ha mentettünk) a jelentéshez hozzá tudjuk rendelni a Query1-et. Tegyük ezt meg:



16. Helyezzük el a szükséges adatokat tartalmazó elemeket a jelentésünkön: Szükségünk lesz egy QRBand elemre, mely a fejléce lesz a számlánknak. Helyezzük ezt el a jelentés tetején. Helyezzünk el benne egy QRLabel-t, ami tartalmazza a SZÁMLA feliratot a Band tetején középen. Ízlés szerint formázzuk a szokott módon.
17. Helyezzünk a számla feliratunk alá egy QRDBText-et, a dataset legyen a Query, a datafield pedig legyen a Számla_az. Helyezzünk el két QRShape objektumot a mező alá úgy, hogy a rendelkezésre álló területet félig foglalják el:

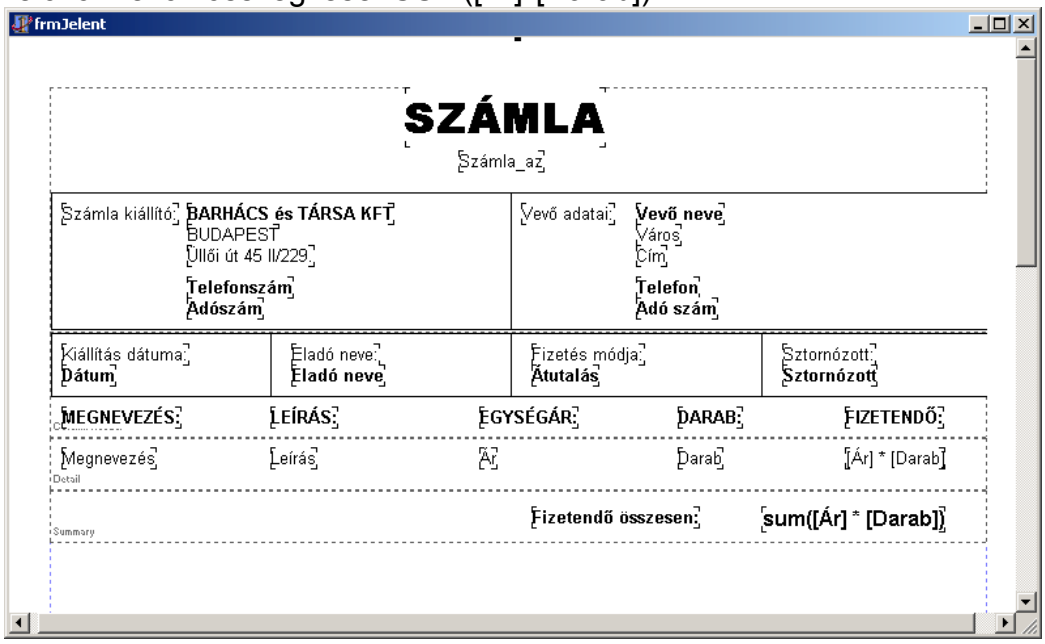


Ebbe a két objektumba kerülnek a fejléc adatok, a cégünk statikus adatai címkékbe (QRLabel), míg a vevő adatai QRDBText objektumokba. Vegyük fel a mezőket, és töltsük ki az eladó adatait találomra, a vevő adatait pedig a QRDBText-ek segítségével, a megfelelő mezők hozzárendelésével.
Ezt követően egy újabb tagoló elmere (QRBand) lesz szükségünk, aminek tulajdonságát (BandType) állítsuk rbColumnHeader-re. Ezt az elemet használjuk a számlaadatok fejrészéhez, vagyis ide kerülnek a fejlécadatok a termékeinkkel kapcsolatban, valamint a számla további szükséges adatai. Itt is használjuk a QRLabel, és a QRDBText objektumokat, valamint a QRShape-et:



Természetesen a Shap-ek alatt lévő elemek csupán label objektumok, a kiemelt elemeink a Shapben felette viszont QRDBText-ek. Jöjjön egy újabb szint, vagyis egy QRBand. Ez állítsuk rbDetail-re. Ne ijedjünk meg, ha nem akarja az újabb Band-et a delphi az előző alatt elhelyezni, ha beállítottuk a Detail tulajdonságot, automatikusan a header alá fog kerülni.

18. Ebben a detail mezőben helyezzük el a címkéknek megfelelő QRDBText objektumokat a megfelelő mezőkhöz rendelve azokat, egészen a Darab-ig.
19. Az utolsó elemünk (a fizetendő) természetesen számított mező lesz majd, így az nem QRDBText, hanem QRExpr (vagyis kifejezés) objektum. Helyezzük el a helyére, majd az objektumkezelőben az Expression tulajdonsághoz írjuk be: [Ár]*[Darab].
20. Maradt még az összegzés, a számla tételeinek összesítése a számlánk végén. Ehhez szükségünk lesz egy újabb Band-re, melyet most állítsunk rbSummary típusra. Helyezzünk el egy Labelt a jobb szélen, majd mellette egy újabb számított mezőt (QRExpr), aminél az Expression most legyen az előző mezők összegzése: SUM([Ár]*[Darab]).



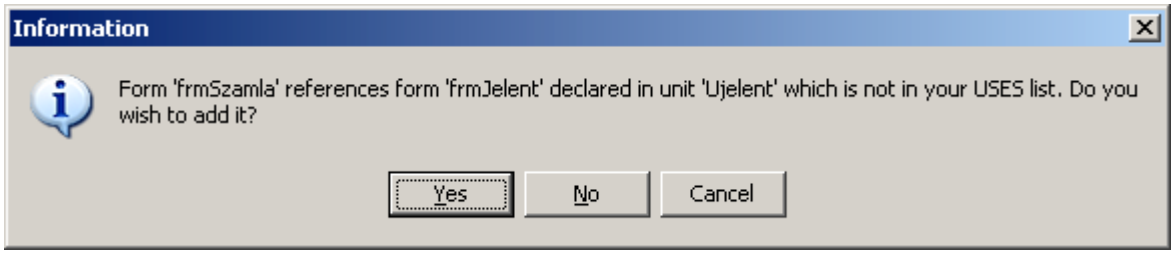
21. Ha szeretnénk, még elhelyezhetünk az oldalak aljára egy címkét, ekkor egy újabb Band-re lesz szükségünk, BandType: rbPageFooter. A címkére (QRLabel) pedig akármit írhatunk, mondjuk a készítő nevét.

Természetesen számlánkat időnként ellenőrizhetjük (jobb gomb, és Preview), de sok örömünk nem lesz benne, hisz az adatbázisból származó adataink megjelenítéséért felelős lekérdezés nem aktív, vagyis csupán a statikus szövegeinket tekinthetjük meg, de azokat is csak ott, ahol nem kötődnek az adatbázis mezőkhöz. Nézzük hogyan jeleníthetőek meg az elkészült jelentésünkön az adatok:

22. Lépjünk vissza a főformunkra (frmSzamla), majd keressük meg a korábban már elkészített kódunkat a kombi panelnél. Ezt kell kiegészítenünk a jelentésünk futtatásához szükséges kóddal, majd mentést kérnünk:

```
procedure TfrmSzamla.DBLookupComboBox1CloseUp(Sender: TObject);
begin
 Query1.params[0].Text:=dblookupcombobox1.text;
 query1.Close;
 query1.open;
 frmJelent.Quickrep1.preview;
end;
```

23. Ekkor ismét jön a szokásos figyelmeztetés:



Természetesen igen a válasz, s ha mentés után futtatunk, meg kell jelenjen a képernyőn elkészült jelentésünk.
A megjelent képen nyilván szemmel látható néhány finomítani való, például a mezők igazítása egymáshoz képest, a megjelenítendő adatokhoz képest, a pénznem formátum megjelenítése, ha a termék neve hosszú, akkor átlóg az egységár alatt, a fizetés módjának megjelenítéséhez szükségünk lenne egy logikai függvényre, így szövegesen írhatnánk ki, hogy készpénzes, vagy átutalásos a számla, és így tovább… Ezek a csinosítások természetesen az idő függvényében megvalósíthatóak.

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Az SQL utasításon belüli paraméter értékadásához a következő formát használhatjuk?
   a., :=
   b., =
   c., =:
2. A jelentés varázslóval melyik jeletés készíthető el?
   a., List Report
   b., Master Detail Report
   c., Query Report
3. Jelentésünk megjelenítéséhez melyik utasítás használható?
   a., QuickRep1.PrintPreview
   b., QuickRep1.Executive
   c., QuickRep1.Preview
4. Mire alkalmas a QRExpr komponens?
   a., Csoportszintek létrehozására
   b., Szűrések megjelenítésére
   c., Számított mezők létrehozására

II.

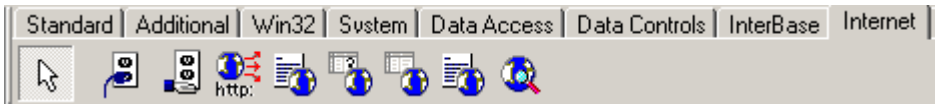
KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. Az access adatbáziskezelő lehetőséget biztosít Interbase formátumú mentésre.
   igaz
   hamis
2. A QRSysData objektum rendszeradatok beállítását teszi lehetővé.
   igaz
   hamis
3. A QRBand csoportosításra, szintek létrehozására alkalmas.
   igaz
   hamis
4. Egy jelentésben is tudunk számított mezőkkel dolgozni.
   igaz
   hamis
5. A QRShape objektum a tagolásokat, csoportosításokat könnyíti meg a jelentésben.
   igaz
   hamis

A Delphi és a Web.

A Delphi tippek és trükkök fejezetében érintőlegesen már foglalkoztunk a helyi gépre telepített böngésző indításával, a levelezés kezdeményezésével, valamint az IE böngésző elrejtésével, és megjelenítésével. Nem véletlenül tettünk különbséget a gépre telepített alapértelmezett böngésző, és az Internet Explorer között, hisz napjainkban egyre gyakoribb a windows környezetben futtatott egyéb böngészők használata.
Természetesen a Delphi is támogatja az Internet oldalú fejlesztéseket, melyhez külön komponenseket találunk:

Az Internet komponenspaletta:



ClientSocket: TCP/IP alapú kliens
ServerSocket: TCP/IP alapú szerver
WebDispacher: Adatok konvertálása Web felületre
PageProducer: HTML konvertálása a kliens gépre
QueryTableProducer: Egy paraméterezhető lekérdezés megjelenítéséhez
DataSetTableProducer: Egy tábla, lekérdezés adatainak megjelenítéséhez
DataSetPageProducer: HTML kód generálása az adatokból
WebBrowser: Egy beépített böngésző elhelyezése a Delphi alkalmazáson belül

Weboldal indítása (WEB)

Amit ebben a fejezetben szeretnénk, az egy saját böngésző készítése. Kezdjük a legelején, hozzuk létre a WEB alkönyvtárat.

1. Indítsuk a Delphit, a form name: frmWeb, Caption: Barhács és társa böngésző. A WindowState: wsMaximized (teljes képernyőn szeretnénk futtatni), BorderIcons/biMaximized: False (nem szeretnénk, ha a felhasználó ezt megváltoztatná).
2. Save All: Uweb, Pweb a WEB mappába.
3. Helyezzünk el most egy WebBrowser elemet az Internet palettáról, maradjon minden beállítás alapértelmezett, csak az Align legyen alClient. Ezzel a formon belül teljes képernyőn fog megnyílni a kívánt lap.
4. A Win32 palettáról helyezzünk el egy ToolBar elemet, majd csökkentsük a méretét (Height: 25)
5. A Standard palettáról tegyünk még fel a formra egy ComboBox-ot, Name:cim, Width: 350. Kattintsunk az Items (TStrings)-re, majd adjunk meg 3 Internet címet (www.barhacs.hu ; http://tibusoft.srv.hu ; ftp.barhacs.hu)
   Helyezzük el a ComboBox-ot a ToolBar-on középtájt, ha ekkor eltűnik, akkor menjünk az egérrel a ToolBar szürke területére, és a jobb egérgomb helyi menüjéből kérjük a Send To Back (háttérbe küldés) parancsát, így a combobox előtérben marad.
6. Lépjünk most át a ComboBox Events fülére, majd keressük meg az OnChange eseményt. A kódszerkesztőbe pedig írjuk be a következő parancsot:

```
procedure TfrmWeb.cimChange(Sender: TObject);
begin
webbrowser1.Navigate(cim.text);
end;
```

A kódunk beállítja a webböngészőnk címének az általunk kiválasztott értéket a címnek nevezett kombi panelről. Természetesen a panelbe felvihetünk egyéb címeket is, de mivel az OnChange eseményt használjuk, így a beírás közben már elkezdi keresni böngészőnk a gépelés alatt álló szöveget. Ez önmagában nem probléma (sőt, tulajdonképpen megtáltosodik tőle az internetünk), de ha csupán részcímet írunk be, akkor viszonylag gyorsan kapjuk majd meg a „lap nem érhető el” típusú hibaüzenetet. Munkánkat akár az OnEnter eseményben is elvégezhettük volna, döntsük el melyik szimpatikusabb.
A következő lépés a léptető gombok elhelyezése:

7. Menjünk a ToolBar1 elemünkhöz, és jobb egérgombbal kérjük kétszer a New Button lehetőséget, majd egyszer egy New Separator-t, egy New Buttont, újabb Separatort, és végül még egy gombot, valahogy így:

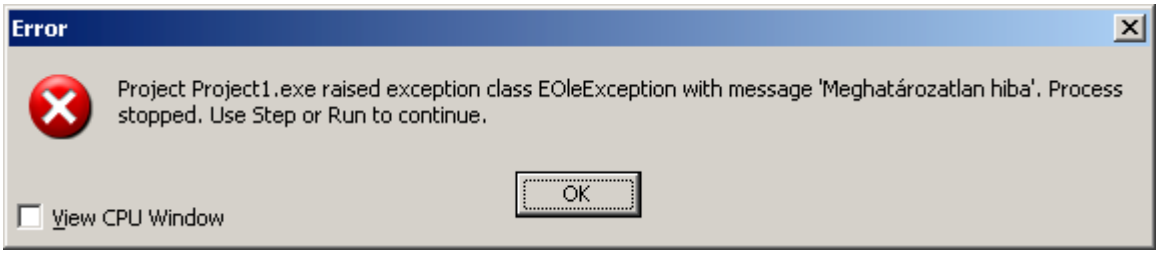
8. A gombokhoz (mivel egy egységet alkotnak a ToolBar-on) csak együtt adhatunk képeket (bmp, vagy ico). Válasszunk ki egy ImageList elemet (Win32), és kattintsunk rajta duplán a bal egérgombbal. A megjelenő szerkesztőben keressük meg a gombjainkhoz leginkább illő ikonokat, (ha nem tetszik egyik sem hozzuk őket létre), majd rendeljük hozzá a ToolBar-hoz az ImageList1-et. Látni fogjuk, hogy a képek felhelyezésüknek sorrendjében jelennek majd meg a gombjainkon. Írjuk meg a hozzájuk tartozó kódokat:
9. Válasszuk ki az első button-t, kattintsunk rá duplán. Ez lesz a visszalépésért felelős gombunk, a kódja pedig (vigyázat veszélyes!):

```
procedure TfrmWeb.ToolButton1Click(Sender: TObject);
begin
WebBrowser1.GoBack;
end;
```

10. Természetesen a következő gombunk kódja is ennyi lesz nagyjából:

```
procedure TfrmWeb.ToolButton2Click(Sender: TObject);
begin
WebBrowser1.GoForward;
end;
```

Persze az Élet, mint tudjuk nem ilyen egyszerű, ha jártunk már néhány oldalon, akkor lesz előző, és következő oldalunk, melyre szépen reagálni is fog a kód, de abban az esetben, ha indítás után rögtön az előző, vagy esetleg a következő elemre szeretnénk lépni, a már rég látott kis hibaüzenetet fogjuk kapni:



Ebből nyilván a meghatározatlan hiba jelentené a létfontosságú információt, ne csüggedjünk, a gond, hogy még nem jártunk körbe. Nyilván egy egyszerű elágazással megvizsgálható a lapra léptetésnél, hogy létezik-e előző, illetve következő. De mi most haladunk tovább, a hibakezelésre az óra végén lesz majd bőségesen idő. A következő gombunk a nyomtatás lesz. A harmadik (önállóan szeparált gombunkhoz) írjuk be a következő kódot:

```
procedure TfrmWeb.ToolButton4Click(Sender: TObject);
var  vIn, vOut : OleVariant;
begin
WebBrowser1.ControlInterface.ExecWB(OLECMDID_PRINT,
        OLECMDEXECOPT_DONTPROMPTUSER, vIn, vOut);
end;
```

Ahhoz, hogy elkerüljük a véletlen nyomtatásokat, feltétlenül helyezzünk el egy egyszerű párbeszédablakot megerősítéssel a mostani kódunkba, ugyanis egy kattintásra azonnal kezdődik a nyomtatás. (Mondjuk a Háború és Béke online olvasása közben ez némi gondot, ideges kapkodást eredményezhet.)
Maradt még egy gombunk, ismétlésként hívjuk meg a korábbi levelezést:

11. Ha emlékszünk még, akkor a levelezéshez a ShellApi-t fel kell vennünk a USES listába, és egy függvénnyel tudjuk utána szóra bírni az alapértelmezett levelezőprogramunkat.
12. Az utolsó gombra duplát kattintva a megjelenő kódszerkesztőbe írjuk be a következőt:

```
procedure TfrmWeb.ToolButton6Click(Sender: TObject);
begin
ShellExecute(Handle,'open','mailto:tibusoft@freemail.hu?Subject=Kérdésem lenne',
        '','',SW_SHOWDEFAULT);
end;
```

Ha futtatnánk, nem sikerülne a fordítás, hisz a ShellApi-t még nem írtuk be a USES deklarációs listába. Tegyük meg, majd mentsünk, és futtassunk. Elkészült első saját készítésű, egyszerű web böngészőnk.
Egészítsük ezt ki ízlés szerint önállóan a hátralévő időben, törekedjünk a precíz hibakezelésekre!

Ellenőrző kérdések

I.

KÉREM VÁLASSZA KI A HELYES MEGOLDÁST!

1. Melyik nem része a Delphi Internet komponenspalettájának?
   a., PageProducer
   b., WebDispatcher
   c., PageHeader
2. Melyik egy weboldal megjelenítésének legalapvetőbb komponense?
   a., WebBrowser
   b., WebServer
   c., ClientSocket
3. Melyik tulajdonság alkalmas a webcím beállítására a böngészőnkben?
   a., AdressBar
   b., Execute
   c., Navigate
4. Melyik függvényt használhatjuk a levelezés indításához?
   a., SendMessage
   b., WinExec
   c., ShellExecute

II.

KÉREM DÖNTSE EL, HOGY IGAZ, VAGY HAMIS-E AZ ÁLLÍTÁS!

1. A WebBrowser komponens teljesképernyős módra is állítható.
   igaz
   hamis
2. A „Meghatározatlan hiba” ellen nem tudunk védekezni.
   igaz
   hamis
3. A visszalépéshez a „WebRowser1.Back” utasítás használható.
   igaz
   hamis
4. A DataSetPageProducer lekérdezésink megjelenítését teszi lehetővé.
   igaz
   hamis
5. A Navigate függvény webcím átvételét teszi lehetővé.
   igaz
   hamis

Gyakorló feladatsorok.

A Delphi házivizsga gyakorlati része 3 fő témakör köré épül. Az első témakör egy önálló alkalmazás elkészítése, a második egy összetett többablakos alkalmazás fejlesztése, és a harmadik egy adatbázis kezelő alkalmazás elkészítése.
A feladatok természetesen nem azonos súlyúak, de jelen jegyzetünkben nagyjából igyekeztünk az arányokat megtartani, vagyis az esetlegesen könnyebbnek tűnő feladatokhoz kevesebb segítséget tartalmaz a jegyzet, míg az összetettebb feladatok esetében lényegesen körültekintőbbek igyekeztünk lenni, vagyis jóval több segítséget nyújtottunk.
A vizsgázónak a következő szempontok alapján lehetősége van szabadon választani a 3 témakör közül, és azt az egy feladatot elkészíteni, mely számára a legszimpatikusabb.

1. SINGLE DOCUMENT INTERFACE

A feladat, egy önálló egy ablakban futó alkalmazás fejlesztése. Példaként a Windows operációs rendszer beépített alkalmazásait említenénk: Jegyzettömb, Média Player, Pasziánsz, stb.
Amennyiben az alkalmazás témája lehetővé teszi, rendelkezzék saját menüvel, eszköztárral.
Megjelenésében kövesse a Windows konvenciókat, de érdekes egyedi megoldásokat is tartalmazhat.
Legyenek benne hibakezelések, kivételkezelések.
Törekedjünk a felhasználóbarát felület kialakítására.
A feladatot óvatosan körvonalazzuk magunknak, ne akarjunk túl sok funkciót beépíteni, nehogy kifussunk az időből.

2. MULTIPLE DOCUMENT INTERFACE

A feladat, egy többablakos alkalmazás elkészítése, menüvezérelt felületen, a hallgató által eddig készített munkák beépítésével, azok felhasználásával, valamint egy egyszerű, eddig még el nem készített alkalmazás megjelenítésével.
Az alkalmazás menüvezérelt legyen, eszköztárral kell rendelkeznie, tartalmazzon súgót, és saját névjegyet.
Kövesse a Windows konvenciókat, de ez is tartalmazhat érdekes megoldásokat.
Ügyeljünk a kivételkezelésre, tartalmazzon a kód hibakezeléseket.
A feladat megfogalmazás itt is legyen óvatos körültekintő, az idő függvényében haladjunk, ne akarjunk túl sok mindent.

3. ADATBÁZIS KEZELÉS

A feladat, egy három táblás adatbázis kezelő alkalmazás fejlesztése, menüvezérelt felülettel, bővítési, módosítási lehetőségekkel, lekérdezésekkel, fő, és segédűrlapokkal, valamint jelentésekkel.

Megoldások

1. fejezet

I.

1. a.
2. b.
3. c.
4. b.
5. c.
6. c.
7. b.
8. a.

II.

1. hamis
2. hamis
3. igaz
4. igaz
5. igaz
6. igaz

III.

1. Az Edit menüben.
2. 15.
3. Add to project
   Remove from project
   Import Type library
   Add to repository
   View Source
   …
4. Project állomány: DPR.
   Unit állomány: PAS.
   Delphi form: DFM.
   Konfigurációs állomány: CFG.
   Windows erőforrás információk: RES.
   Project beállítások: DOF.
   Átmeneti állományok: DCU.
5. Fejrész.
   Objektum-felügyelő.
   Formszerkesztő.
   Kódszerkesztő.
6. A mentés során a Unit(ok), és a project mentése történik. A többi állomány mentését a gép automatikusan végzi.
7. Objektum-felügyelő, a komponenseink tulajdonságainak beállítását teszi lehetővé vizuális felületen.
8. New, Open, Save, Add to project, Help, Toggle Form/Unit, …

2. fejezet

I.

1. c.
2. c.
3. c.
4. a.
5. b.
6. a.
7. b.
8. b.

II.

1. igaz
2. hamis
3. hamis
4. igaz
5. igaz
6. hamis
7. igaz
8. igaz
9. hamis

III.

1. Az osztályt, jele: T.
2. BorderIcons, Caption, Enabled, Font, Name, Visible, ...
3. OnActivate, OnClick, OnDblClick, OnShow, OnCreate, …
4. Az űrlap felhasználó által történő méretezhetőségét, formáját.
5. Billentyű felengedésekor.
6. UNIT név.
   Interface
   Uses lista (használni kívánt Unit-ok)
   Type (típusdefiníciók)
   Var (változó deklarációk)
   Implementation (az általunk megírt függvények, eljárások)
   End. (A kód lezárása)

3. fejezet

I.

1. b.
2. c.
3. b.
4. a.
5. b.
6. a.
7. c.

II.

1. hamis
2. hamis
3. igaz
4. hamis
5. hamis
6. igaz
7. igaz
8. igaz

III.

1. Egy sor komment: //
   Összetett kommentek:
   (*…*)
   {…}
2. ShowMessage, MessageDlg, MessageBox.
3. MessageDlg('Üzenet', ablak típusa, [Gomb1, Gomb2, …],súgó).
4. MainMenu komponens elhelyezése, Caption kitöltése, esetleges almenük elkészítése, a menükhöz tartozó kódok beírása.
5. Egy ablak bezárásakor aktivizálódó esemény, melyben a bezárást korlátozó kódot adhatunk meg.
6. Egyszerű szöveges üzenetek megjelenítésére.

4. fejezet

I.

1. c.
2. c.
3. c.
4. c.
5. a.

II.

1. hamis
2. hamis
3. igaz
4. hamis
5. hamis
6. igaz
7. hamis

III.

1. Raise, Try…Except. Try…Finally.
2. Állományokkal való műveletek során használjuk, a Try rész tartalmazza az állományra vonatkozó utasításainkat, a Finally részben pedig a zárási utasítások lesznek.
3. StrToInt, StrToFloat.

4. Az esetleges hibák kiküszöbölése annak érdekében, hogy hiba esetén ne kerüljön ki a vezérlés a program kezéből.
5. Futás idejű hibák (a programozón kívül álló okok miatt).
   Programozói hibák (szintaktikai, szemantikai).

5. fejezet

I.

1. b.
2. c.
3. b.
4. a.
5. a.
6. c.
7. b.
8. b.

II.

1. hamis
2. hamis
3. hamis
4. igaz
5. hamis
6. hamis
7. igaz
8. hamis
9. igaz
10. hamis

6. fejezet

I.

1. a.
2. b.
3. b.
4. b.

II.

1. igaz
2. igaz
3. hamis
4. igaz
5. igaz

III.

1. Az aktuális munkánk valamely elemének eltávolítására használhatjuk.

2. Például: üres a beviteli mező, értelmetlen adatot ad meg a felhasználó, nem háromszöghöz tartozó adattal akar dolgozni, negatív értékek, nullával való osztás, stb.
3. A legegyszerűbben annak nevével.
4. Képek megjelenítése az űrlapokon.
5. ListBox esetén az Items tulajdonsággal tölthetjük fel listánkat.

7. fejezet

I.

1. Memóriaterület felszabadítására.
2. Keretbe foglalja többablakos alkalmazásunkat.
3. Mert automatikusan létrehozza őket a Delphi indításkor.
4. Olyan események elhelyezésére, melyeket az objektum aktiválásakor (vagyis ha rákerül a fókusz) szeretnénk végrehajtatni.
5. Multiple Document Interface, többablakos alkalmazás. Az alkalmazásnak tartalmaznia kell legalább egy MDI formot (anyaablakot), melyben a Child (gyermekablakok) megjelenhetnek majd.

8. fejezet

I.

1. c.
2. b.
3. b.
4. c.

II.

1. hamis
2. igaz
3. igaz
4. igaz
5. hamis

9. fejezet

I.

1. b.
2. c.
3. c.
4. c.

II.

1. igaz
2. igaz
3. hamis
4. hamis
5. igaz

10. fejezet

I.

1. b.
2. c.
3. a.
4. b.

II.

1. hamis
2. hamis
3. igaz
4. igaz
5. igaz

11 fejezet

I.

1. b.
2. b.
3. b.
4. c.

II.

1. igaz
2. hamis
3. hamis
4. hamis
5. hamis

12 fejezet

I.

1. c.
2. a.
3. c.
4. c.

II.

1. igaz
2. hamis
3. igaz
4. igaz
5. igaz

13 fejezet

I.

1. c.
2. a.
3. c.
4. c.

II:

1. igaz
2. hamis
3. hamis
4. hamis
5. igaz

Tartalomjegyzék