

## 2.10. Objektumok az Object Pascal-ban

1. Négyzet és kocka adatainak számítása objektumokkal [NegyzetKockaPr](#)
2. Kör adatainak számítása objektumokkal, a rendszerbeállítástól függő tizedesjellel [KorPropPr](#)
3. Származtatott háromdimenziós vektorosztály [Vektorok](#)
4. Vektorosztály polár-koordinátákkal [PVektorok](#)
5. Származtatott, négyzetet és téglalapot reprezentáló osztályok [Geometria](#)
6. Négyzet, téglalap, illetve kör kirajzolása, mozgatása [GrafGeom](#)
7. Pontok, négyzetek, téglalapok és körök objektumlistában [GrafGeomList](#)
8. Nevek keverése és sorba rendezése [SKeveres](#)



Számítsuk ki a négyzet területét, kerületét és a kocka felszínét, térfogatát. A számításához definiáljunk objektumot, valamint származtatott objektumot. (*NegyzetKockaPr*)

A *Negyzet* modul a *TNegyzet* osztály deklarációját, továbbá a terület és a kerület számításához szükséges metódusokat tartalmazza. A számol metódus hívása után a *Get\_ered1* a területet és a *Get\_ered2* a kerületet szolgáltatja. Az *a*, *ered1* és *ered2* mezők az oldalhosszat és a *szamol* hívása után a területet és a kerületet tartalmazzák.

```
unit Negyzet;

interface

type
  TNegyzet = class           // A TNegyzet osztály
    a, ered1, ered2 : real;
    procedure Init(a0:real);
    procedure szamol; virtual;
    function Get_ered1: real;
    function Get_ered2: real;
  end;

implementation

  procedure TNegyzet.Init;
  begin
    a := a0;                // Inicializáláskor az oldal mérete
  end;
  procedure TNegyzet.Szamol;
  begin
    ered1 := a * a;          // A terület
    ered2 := 4*a;            // A kerület
  end;
  function TNegyzet.Get_ered1:real;
  begin
    result := ered1;
  end;
  function TNegyzet.Get_ered2:real;
  begin
    result := ered2;
  end;
end;
```

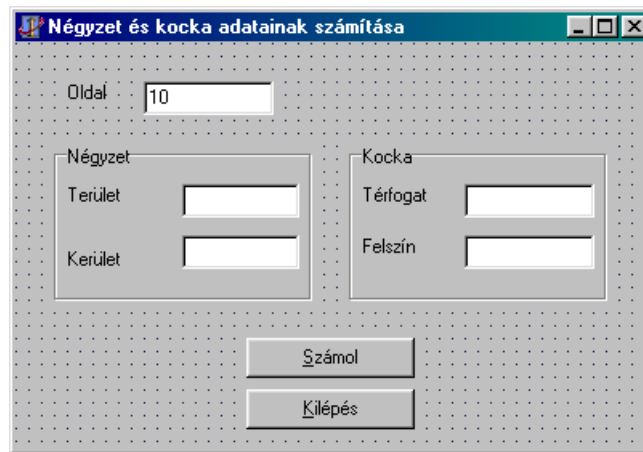
A *Kocka* modul a *TKocka* definícióját tartalmazza. A *TKocka* a *TNegyzet* leszármazottja és a *szamol* átdefiniálásával a *Get\_ered1* a térfogatot és a *Get\_ered2* a felületet adja vissza.

```
unit Kocka;

interface
uses Negyzet;
type
  TKocka = class(TNegyzet)
    procedure szamol; override;
  end;

implementation
  procedure TKocka.Szamol; // Az átdefiniált számoló metódus
  begin
    ered1 := a * a * a;    // Térfogat
    ered2 := 6*a*a;        // Felület
  end;
end.
```

Tervezzük meg programunk felületét az alábbiak megfelelően!



A formon globálisan deklaráljuk az oldalhossz változóját ( $a$ ) és a kocka és négyzet objektumtípusokat.

```
var
  a_oldal : real;
  n: TNegyzet;
  k: TKocka;
```

A form létrehozásakor elkészítjük az objektumokat is.

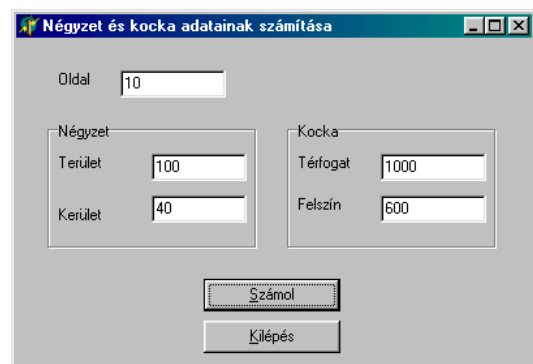
```
procedure TForm1.FormCreate(Sender: TObject);
begin
  n := TNegyzet.Create;
  k := TKocka.Create;
end;
```

Kilépéskor felszabadítjuk a lefoglalt erőforrásokat:

```
procedure TForm1.KilepesClick(Sender: TObject);
begin
  n.Free;
  k.Free;
  Close;
end;
```

A *Számol* gomb megnyomásakor inicializáljuk objektumainkat, kiszámoljuk az adatokat és megjelenítjük. A hibás adatbevitelt kivételkezeléssel szűrjük ki.

```
procedure TForm1.SzamolClick(Sender: TObject);
begin
  try
    if (edit1.Text) <> '' then
    begin
      a_oldal := StrToFloat(Edit1.Text);
      n.Init(a_oldal);
      k.Init(a_oldal);
      n.Szamol;
      k.Szamol;
      Edit2.Text := FloatToStr(n.Get_ered1);
      Edit3.Text := FloatToStr(n.Get_ered2);
      Edit4.Text := FloatToStr(k.Get_ered1);
      Edit5.Text := FloatToStr(k.Get_ered2);
    end;
  except
    Edit1.Text := '0';
    Beep;
    ShowMessage('Számítási hiba');
  end;
end;
```





Készítsünk olyan alkalmazást, amely a megadott sugarú kör területét és kerületét objektum tulajdonságainak felhasználásával számolja. A program adatbevétele egyaránt működjön tizedes pont és tizedes vessző esetén! (*KorPropPr*)

A kör adatainak számításához a *Kor* modulban deklarált *Tkor* osztályt használjuk. A kör sugara (*sugar*), kerülete (*kerulet*) és területe (*terulet*) egyaránt tulajdonság. Az adatok tárolására szolgáló belső adatmezők az *r*, a *ter* és a *ker*. A sugár írható olvasható (a *SetR* és *GetR* metódusokkal), az utóbbi kettő azonban csak olvasható (*GetTer* és *GetKer* metódusok).

```
unit Kor;

interface
type
  Tkor = class
  protected
    r: real;
    ter, ker: real;
  procedure SetR(r: real);
  function GetR: real;
  function GetTer: real;
  function GetKer: real;
  public
    constructor Create(r: real = 0);
    property sugar: real read GetR write SetR;
    property terulet: real read GetTer;
    property kerulet: real read GetKer;
  end;

implementation

// Metódusok
constructor Tkor.Create;
begin
  SetR(r)
end;

// A sugar tulajdonság elérését biztosító metódusok
procedure Tkor.SetR;
begin
  self.r := r;
end;

function Tkor.GetR;
begin
  result := r;
end;

// A terület lekérdezése
function Tkor.GetTer: real;
begin
  result := sqr(r) * pi;
end;

// A kerület lekérdezése
function Tkor.GetKer: real;
begin
  result := 2 * r * pi;
end;

end.
```

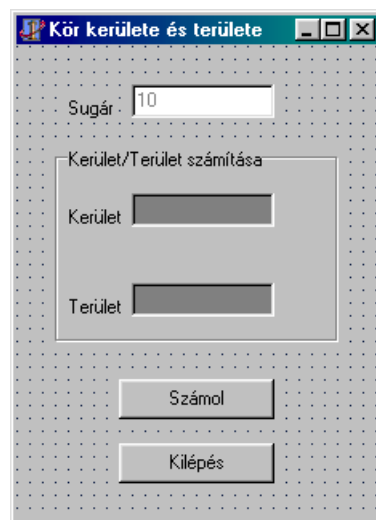
Tervezzük meg a felhasználói felületet! A sugár megadására az *Edit1* szövegszerkesztőt-, az adatok megjelenítésére az *Edit2* és *Edit3* szövegszerkesztőket használjuk. Az utóbbi kettő csak olvasható és színezésük is eltérő.

A számításhoz a *Szamol* nyomógombot, kilépéshez a *Button2* nyomógombot használjuk.

A form létrehozásakor elkészítjük a globálisan deklarált *k* objektumot a *Kor* modul segítségével.

```
uses Kor;
var
  k : Tkor;

procedure TForm1.FormCreate(Sender: TObject);
begin
  k:= Tkor.Create;
end;
```



Kilépéskor gondoskodunk a felszabadításról is.

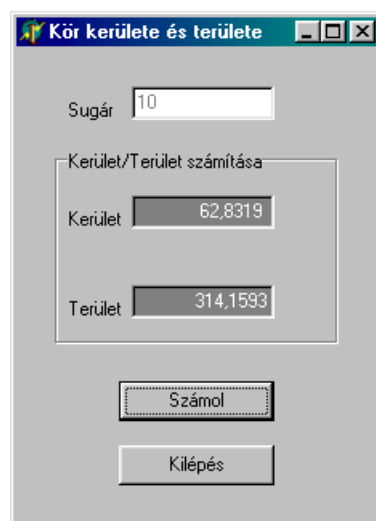
```
procedure TForm1.Button2Click(Sender: TObject);
begin
  k.free;
  Application.Terminate;
end;
```

Ha az adatmegadó szövegszerkesztő tartalma megváltozik, töröljük az eredményablakokat.

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  Edit2.clear;
  Edit3.clear;
end;
```

A számolás során a hibás adatmegadást kivételkezeléssel szűrjük ki. A *DecimalSeparator* karakter tartalmazza a rendszer tizedes jelét és ennek értékét is figyelembe vesszük.

```
procedure TForm1.SzamolClick(Sender: TObject);
begin
  if Edit1.text <> '' then
  begin
    try
      if pos('.',edit1.text)<>0 then
        DecimalSeparator:= '.'
      else
        DecimalSeparator:= ',';
      k.sugar:= StrToFloat(edit1.text);
      Edit2.Text := Format('%20.4f', [k.kerulet]);
      Edit3.Text := Format('%20.4f', [k.terulet]);
    except
      Beep;
      ShowMessage('Hibás sugáradat!');
    end;
  end;
end;
```





A fejezet kétdimenziós vektorából származtassunk háromdimenziós vektor osztályt *TVektor3D* néven! (Vektorok)

A *Vektor2D* modul a 2.10 fejezet kétdimenziós vektorainak osztályát definiálja (*TVektor2D*).

```
unit Vektor2D;

interface
type
    TVektor2D=class
    protected
        // adatmezők
        fx,fy : integer;
    public
        // metódusok
        function LekerX : integer;
        procedure BeallitX(a : integer);
        function LekerY : integer;
        procedure BeallitY(a : integer);
        function VektorToStr : string; virtual;
    published
        constructor Letrehoz(a,b : integer);
        class function HanyDimenzios : integer; virtual;
        // tulajdonságok
        property x : integer read fx write BeallitX;
        property y : integer read fy write BeallitY;
    end;

implementation
uses sysutils;

// Inicializáló metódus
constructor TVektor2D.Letrehoz(a,b : integer);
begin
    fx:=a;
    fy:=b;
end;

// Osztálymetódus
class function TVektor2D.HanyDimenzios : integer;
begin
    result:=2;
end;

// A vektor koordinátáit lekérdező függvények
function TVektor2D.LekerX : integer;
begin
    result:=Self.fx;
end;
function TVektor2D.LekerY : integer;
begin
    result:=fy;
end;

// A vektor koordinátáit beállító eljárások
procedure TVektor2D.BeallitX(a:integer);
begin
    fx:=a;
end;
procedure TVektor2D.BeallitY(a:integer);
begin
    fy:=a;
end;
function TVektor2D.VektorToStr:string;
begin
    result:=format(' (%d,%d) ', [x,y]);
end;

end.
```

A *TVektor3D* a *TVektor2D* leszármazottja. Új adatmező az *fz*, írható (*BeallitZ*), olvasható (*LekerZ*) tulajdonság a *z*. A konstruktor az **inherited** *Letrehoz* hívást kiegészíti a harmadik koordinátával. A dimenziószámnak megfelelően átdefiniáljuk a *Hanydimenzios* metódust (**override**). A *VektorToStr* osztálymetódus elrejt az örökölt virtuális metódust és újjal helyettesíti a **reintroduce** direktíva hatására.

```
unit Vektor3D;

interface
uses Vektor2D;
type
    TVektor3D=class(TVektor2D)
    protected
        // adatmezők
        fz : integer;
    public
        // metódusok
        function LekerZ : integer;
        procedure BeallitZ(a : integer);
        function VektorToStr : string; reintroduce;
    published
        constructor Letrehoz(a,b,c : integer);
        class function HanyDimenzios : integer; override;
        // tulajdonságok
        property z : integer read LekerZ write BeallitZ;
end;

implementation
uses sysutils;

// Inicializáló metódus
constructor TVektor3D.Letrehoz(a,b,c : integer);
begin
    inherited letrehoz(a,b);
    fz:=c;
end;

// Osztálymetódus
class function TVektor3D.HanyDimenzios : integer;
begin
    result:=3;
end;

// A vektor z-koordinátáját lekérdező függvény
function TVektor3D.LekerZ : integer;
begin
    result:=Self.fz;
end;

// A vektor z-koordinátáját beállító eljárás
procedure TVektor3D.BeallitZ(a:integer);
begin
    fz:=a;
end;

// A vektor sztringgé alakítása
function TVektor3D.VektorToStr:string;
begin
    result:=format(' (%d,%d,%d) ', [x,y,z]);
end;

end.
```

A főprogramban használjuk a *Vektor2D* és *Vektor3D* modulokat. A *v2* és *v23* kétdimenziós, míg a *v3* háromdimenziós vektorok.

```
{ $APPTYPE CONSOLE }
uses Vektor2D, Vektor3D;
var
  v2, v23 : TVektor2D;
  v3       : TVektor3D;
```

Első lépésben a *v2* kétdimenziós vektort létrehozuk és kiírjuk adatait.

```
writeln(TVektor2D.HanyDimenzios, '-dimenzios vektor:');
v2:=TVektor2D.Letrehoz(100,200);
with v2 do
begin
  x:=x+25;
  y:=y-25;
  writeln(VektorToStr);
end;
```

Hasonlóan járunk el a *v3* háromdimenziós vektorral.

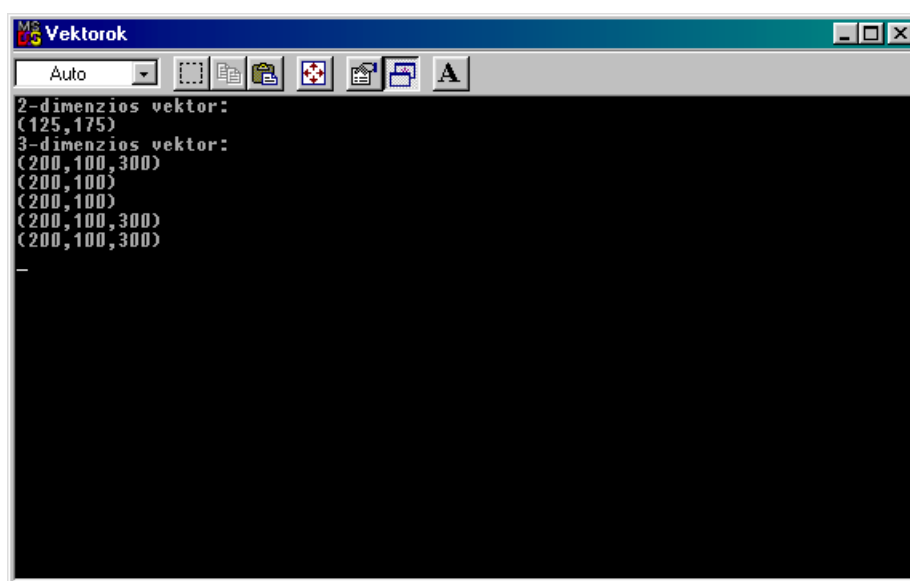
```
writeln(TVektor3D.HanyDimenzios, '-dimenzios vektor:');
v3:=TVektor3D.Letrehoz(100,200,300);
v3.x:=v3.x*2;
v3.y:=v3.y div 2;
v3.z:=v3.x+v3.y;
writeln(v3.VektorToStr);
```

A *v23* kétdimenziós vektor típusú változó koordinátáit a háromdimenziós *v3* alapján értékadással kapjuk, így ez referenciaként a *TVektor2D* és a *TVektor3D* típusú objektumokra is hivatkozhat. Ennek bemutatására az adatokat kétféle típuskonverzióval is, kétdimenziós vektorként és háromdimenziós vektorként is visszaírjuk.

```
v23:=v3;
writeln(TVektor2d(v23).VektorToStr);
writeln((v23 as TVektor2D).VektorToStr);
writeln(TVektor3d(v23).VektorToStr);
writeln((v23 as TVektor3D).VektorToStr);
```

Nem feledkezhetünk meg az objektumok felszabadításáról sem.

```
v2.free;
v3.free;
```







Tegyük alkalmassá az előző feladatban kialakított osztályokat síkbeli polár-koordináták használatára! (*PVektorok*)

A *Vektor2D* modulban kiegészítjük a [Vektorok](#) kétdimenziós vektor osztályát az *a* szöveget jellemző- és a *d* hossz tulajdonsággal. Az írásra és olvasásra szolgáló metódusok a *LekerA*, *LekerD*, *BeallitA*, *BeallitD*. A *PVektorToStr* metódus a vektor polár-koordinátás adatainak kiírását szolgálja. A *Polar* metódus polár-koordinátákból kiszámítja a Descartes-koordinátákat. Az osztály deklarációja:

```
type
  TVektor2D=class
  protected
    // adatmezők
    fx,fy : real;
  public
    // metódusok
    function LekerX : real;
    procedure BeallitX(x : real);
    function LekerY : real;
    procedure BeallitY(y : real);
    function LekerA : real;
    procedure BeallitA(a : real);
    function LekerD : real;
    procedure BeallitD(d : real);
    function VektorToStr : string; virtual;
    function PVektorToStr : string; virtual;
    procedure Polar(d,a : real);
  published
    constructor Letrehoz(a,b : real);
    constructor LetrehozPolar(d,a :real);
    class function HanyDimenzios : integer; virtual;
  // tulajdonságok
  property x : real read fx write BeallitX;
  property y : real read fy write BeallitY;
  property a : real read LekerA write BeallitA;
  property d : real read LekerD write BeallitD;
end;
```

A megvalósítás során szükségünk lesz a fok-radián átszámító konstansra:

```
const FokRad:real=180/PI;
```

A továbbiakban csak azokat a metódusokat listázzuk, amelyek kiegészítik a [Vektorok](#) metódusait.

```
// Inicializáló metódus
constructor TVektor2D.LetrehozPolar(d,a :real);
begin
  Polar(d,a);
end;

// A vektor koordinátáit lekérdező függvények
function TVektor2D.LekerA : real;
begin
  if fx<>0 then
    result:=arctan(fy/fx)*FokRad
  else
    result:=90;
end;

function TVektor2D.LekerD : real;
begin
  result:=sqrt(sqr(fx)+sqr(fy));
end;
```

```

// A vektor koordinátáit beállító eljárások
procedure TVektor2D.BeallitA(a:real);
begin
    Polar(self.d,a);
end;

procedure TVektor2D.BeallitD(d:real);
begin
    Polar(d,self.a);
end;

// A vektor sztringgé alakítása
function TVektor2D.PVektorToStr:string;
begin
    result:=format(' (%.0f<%.0f) ', [d,a]);
end;

// A vektor beállítása polár-koordináták segítségével
procedure TVektor2D.Polar(d,a:real);
begin
    fx:=d*cos(a/fokrad);
    fy:=d*sin(a/fokrad);
end;

```

A háromdimenziós térben henger koordináta-rendszert használunk, azaz egy harmadik  $z$  koordinátával egészítjük ki a tulajdonságokat. Így a Vektor3D modul teljes egészében megegyezik a [Vektorok](#) példában megismertekkel.

A főprogram is a [Vektorok](#) példához hasonló, csak kiírjuk a polár-koordinátákat is.

```

program PVektorok;
{$APPTYPE CONSOLE}
uses Vektor2D, Vektor3D;
var
    v21, v22 : TVektor2D;
    v3       : TVektor3D;

begin
    writeln(TVektor2D.HanyDimenzios,'-dimenzios vektor:');
    v21:=TVektor2D.LetrehozPolar(100,30);
    with v21 do
        begin
            write(PVektorToStr+#9);
            writeln(VektorToStr);
        end;
    v21.free;

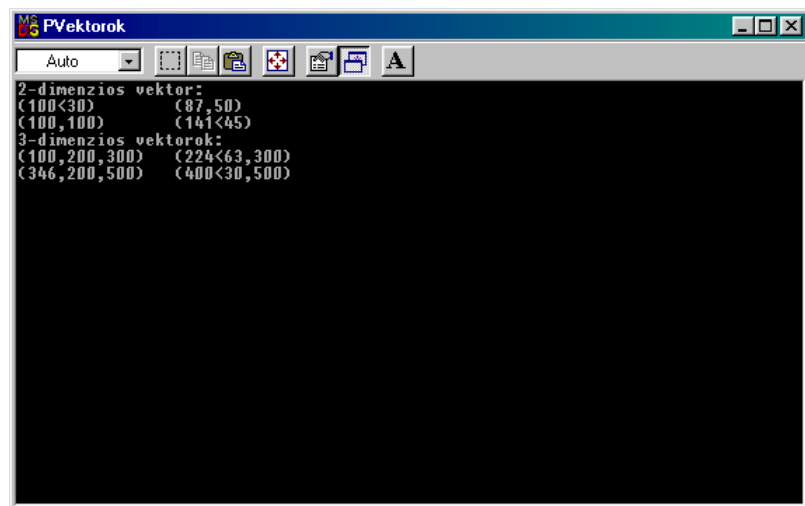
    v22:=TVektor2D.Letrehoz(100,100);
    with v22 do
        begin
            write(VektorToStr+#9);
            writeln(PVektorToStr);
        end;
    v22.free;

    writeln(TVektor3D.HanyDimenzios,'-dimenzios vektorok:');
    v3:=TVektor3D.Letrehoz(100,200,300);
    write(v3.VektorToStr+#9);
    writeln(v3.PVektorToStr);
    v3.free;

    v3:=TVektor3D.LetrehozPolar(400,30,500);
    write(v3.VektorToStr+#9);
    writeln(v3.PVektorToStr);
    v3.free;

    readln;
end.

```





## A 2.10 fejezet *TPozicio* osztálya

```
program Geometria;
{$APPTYPE CONSOLE}
type
// A TPozicio osztaly
TPozicio=Class
    protected
        x,y : integer;
    public
        constructor Create(a,b:integer);
end;
```

A konstruktorban a koordináták értéket kapnak.

```
    constructor TPozicio.Create(a,b:integer);
begin
    inherited create;    // A TObject ós inicializálása, elhagyható
    x:=a; y:=b;
end;
```

A *TPont* osztály (a *TPozicio* leszármazottja) is megfelel a 2.10 fejezetben bemutatottnak, a pontnak a konstruktorban beállítható színe is van.

```
// A TPont osztaly
type
TPont=Class(TPozicio)
    protected
        c : longint;
    public
        constructor Create(px,py:integer;szin:longint);
end;

    constructor TPont.Create(px,py:integer; szin:longint);
begin
    inherited Create(px,py);
    c:=szin;
end;
```

A 2.10. fejezetben láttuk, hogy a *TKor* osztály a *TPont* leszármazottjaként a konstruktorban megadott sugárral is rendelkező objektumok készítéséhez használható.

```
// A TKor osztaly
type
TKor=Class(TPont)
    protected
        r : integer;
    public
        constructor Create(kpx,kpy:integer;
                           szin:longint; sugar:integer);
end;

    constructor TKor.Create(kpx,kpy:integer;
                           szin:longint; sugar:integer);
begin
    inherited Create(kpx,kpy,szin);
    r:=sugar;
end;
```

Származtassuk a *TNegyzet* osztályt a *TPont*-ból úgy, hogy a négyzet oldala is szerepeljen az adatok között!

```
// A TNegyzet osztaly
type
  TNegyzet=Class (TPont)
    protected
      a : integer;
    public
      constructor Create(kpx,kpy:integer;
                        szin:longint; oldal:integer);
end;

constructor TNegyzet.Create(kpx,kpy:integer;
                          szin:longint; oldal:integer);
begin
  inherited Create(kpx,kpy,szin);
  a:=oldal;
end;
```

A *TTeglalap* osztály a *TNegyzet* leszármazottjaként a két oldalt adatként tartalmazza. Az egyik oldalt az örökölt mező, míg a másikat az osztály új mezeje hivatott tárolni.

```
// A TTeglalap osztaly
type
  TTeglalap=Class (TNegyzet)
    protected
      b : integer;
    public
      constructor Create(kpx,kpy:integer;
                        szin:longint; a,b:integer);
end;

constructor TTeglalap.Create(kpx,kpy:integer;
                          szin:longint; a,b:integer);
begin
  inherited Create(kpx,kpy,szin,a);
  self.b:=b;
end;
```

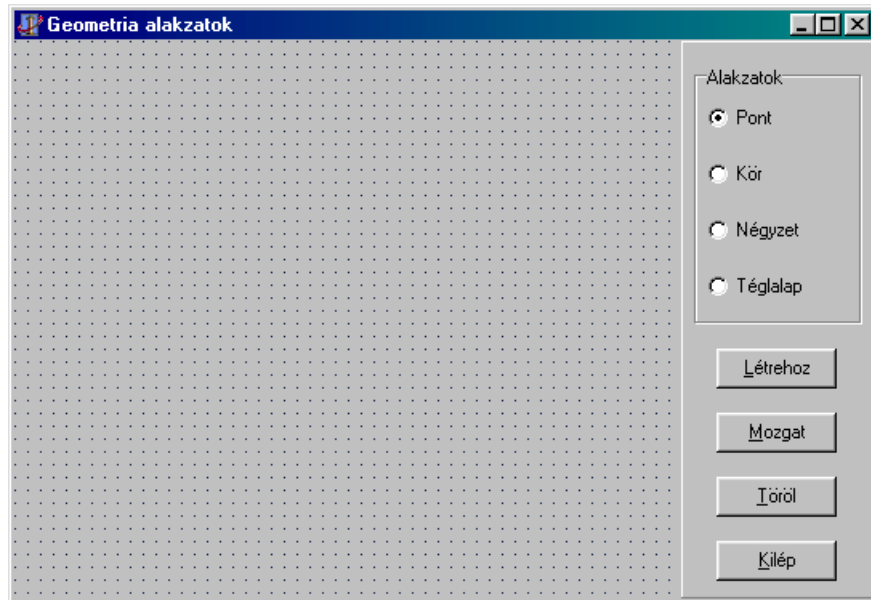
A főprogramban a *TKor* típusú *k* és a *TTeglalap* típusú *t* változókat deklaráljuk, majd létrehozuk és rögtön fel is szabadítjuk az objektumokat.

```
var
  k    : TKor;
  t    : TTeglalap;
begin
  k:=TKor.Create(100,200,14,50);
  t:=TTeglalap.Create(10,20,30,40,50);

  t.Free;
  k.Free;
  readln;
end.
```

A *Geometria* feladatban csak bemutattuk, hogyan lehet geometriai alakzatokat objektumosztályokkal modellezni. Bemutadjuk azt is, hogyan használhatjuk az objektumokat a grafikus megjelenítés során. A feladat megoldása során Windows alkalmazást készítünk és az alkalmazás ablakában rajzolunk is.

Első lépésben tervezzük meg a program felhasználói felületét!



Az objektumok létrehozására, mozgatására és törlésére a *btnLetrehoz*, *btnMozgat* és *btnTorol* nyomógombokat használjuk. A létrehozott alakzatok típusának meghatározásához az *rgAlakzat* választógomb csoportot használjuk. Gondoskodunk majd a programból történő kilépésről (*btnKilep*) is!

```
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    rgAlakzat: TRadioGroup;
    btnLetrehoz: TButton;
    btnMozgat: TButton;
    btnTorol: TButton;
    btnKilep: TButton;
    procedure btnLetrehozClick(Sender: TObject);
    procedure btnTorolClick(Sender: TObject);
    procedure btnMozgatClick(Sender: TObject);
    procedure rgAlakzatClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure btnKilepClick(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

A *Geometria* modulban a *Geometria* feladat alaposztályait megjelenítő metódusokkal egészítjük ki. A *TPozicio* alaposztályban definiáljuk a megjelenítéshez szükséges *Mozgat*, *Rajzol*, *Szine* és *Frissit* metódusokat.

```

type
// A TPozicio absztrakt alaposztaly
TPozicio=Class
  protected
    x,y : integer;
  public
    constructor Create(a,b:integer);
    procedure Mozgat(hovax, hovay:integer);
    procedure Rajzol(szin:TColor); virtual;abstract;
    function Szine:TColor;dynamic; abstract;
    procedure Frissit;
end;

```

Az alaposztály metódusait az **implementation** részben definiáljuk. A konstruktor csak a pozíciót határozza meg.

```

// A TPozicio osztaly metódusai
constructor TPozicio.Create(a,b:integer);
begin
  inherited create;
  x:=a; y:=b;
end;

```

A *Mozgat* és a *Frissit* metódus egyaránt a virtuális absztrakt *Rajzol* metódust hívja, amelyet a konkrét geometriák esetén adunk meg. A *Mozgat* metódus törlésként háttérszínnel (*form1.color*), majd az új pozícióban a megadott színnel (*Szine*) rajzol, ugyanúgy, mint a *Frissit* metódus.

```

procedure TPozicio.Mozgat(hovax, hovay:integer);
begin
  Rajzol(form1.color);
  x:=hovax;
  y:=hovay;
  Rajzol(Szine);
end;

procedure TPozicio.Frissit;
begin
  Rajzol(Szine);
end;

```

A pontok definíciójához és megjelenítéséhez a *TPont* osztály nyújt segítséget.

```

// A TPont osztály
TPont=Class(TPozicio)
  protected
    c : TColor;
  public
    constructor Create(px,py:integer;szin:TColor);
    function Szine:TColor;override;
    procedure Szinez(szin:TColor);
    procedure Rajzol(szin:TColor); override;
end;

```

Az megvalósított metódusok az **implementation** részben találhatók. Létrehozáskor a *Szinez* metódus segítségével a *szin* mezőbe beírjuk a színt és kirajzoljuk a pontot. A *Szine* metódus visszaadja a pont színét.

```

// A TPont osztaly metódusai
constructor TPont.Create(px,py:integer; szin:TColor);
begin
  inherited Create(px,py);
  Szinez(szin);
end;

```

```

function TPont.Szine:TColor;
begin
    result:=c;
end;

procedure TPont.Szinez(szin:TColor);
begin
    c:=szin;
    Rajzol(szin);
end;

```

A tényleges rajzolást a Canvas segítségével végezzük (5. fejezet).

```

procedure TPont.Rajzol(szin:TColor);
begin
    form1.canvas.Pen.Color:=szin;
    form1.canvas.Brush.Style:=bsClear;
    form1.canvas.Moveto(x-5,y);
    form1.canvas.Lineto(x+5,y);
    form1.canvas.Moveto(x,y-5);
    form1.canvas.Lineto(x,y+5);
end;

```

A *TKor* a *TNegyzet* és a *TTeglalap* egyaránt a *TPont* leszármazottja. A körnek sugara, a négyzetnek oldala, míg a téglalapnak oldalai vannak, amiket létrehozáskor állítunk be. Mivel a kört, a négyzetet és a téglalapot a ponttól eltérő módon jelenítjük meg, ezért felüldefiniáljuk a *Rajzol* metódust.

```

// A TKor osztaly
TKor=Class(TPont)
protected
    r : integer;
public
    constructor Create(kpx,kpy:integer;
                      szin:TColor; sugar:integer);
    procedure Rajzol(szin:TColor);override;
end;

// A TNegyzet osztaly
TNegyzet=Class(TPont)
protected
    a : integer;
public
    constructor Create(kpx,kpy:integer;
                      szin:TColor; oldal:integer);
    procedure Rajzol(szin:TColor);override;
end;

// A TTeglalap osztaly
TTeglalap=Class(TNegyzet)
protected
    b : integer;
public
    constructor Create(kpx,kpy:integer;
                      szin:TColor; a,b:integer);
    procedure Rajzol(szin:TColor);override;
end;

```

A metódusok az **implementation** részben:

```

// A TKor osztaly metódusai
constructor TKor.Create(kpx,kpy:integer;
                       szin:TColor; sugar:integer);
begin
    r:=sugar;
    inherited Create(kpx,kpy,szin);
end;

```



```

procedure TKor.Rajzol (szin:TColor);
begin
    form1.canvas.Pen.Color:=szin;
    form1.canvas.Brush.Style:=bsClear;
    form1.canvas.Ellipse(Rect(x-r,y-r,x+r,y+r));
end;

// A TNegyzet osztaly metódusai
constructor TNegyzet.Create(kpx,kpy:integer;
                           szin:TColor; oldal:integer);

begin
    a:=oldal;
    inherited Create(kpx,kpy,szin);
end;

procedure TNegyzet.Rajzol (szin:TColor);
begin
    form1.canvas.Pen.Color:=szin;
    form1.canvas.Brush.Style:=bsClear;
    form1.canvas.Rectangle(Rect(x-a div 2,y-a div 2,
                                x+a div 2,y+a div 2));
end;

// A TTeglalap osztaly metódusai
constructor TTeglalap.Create(kpx,kpy:integer;
                              szin:TColor; a,b:integer);

begin
    self.b:=b;
    inherited Create(kpx,kpy,szin,a);
end;

procedure TTeglalap.Rajzol (szin:TColor);
begin
    form1.canvas.Pen.Color:=szin;
    form1.canvas.Brush.Style:=bsClear;
    form1.canvas.Rectangle(Rect(x-a div 2,y-b div 2,
                                x+a div 2,y+b div 2));
end;

```

A főprogramban használjuk a *Geometria* modult, és deklaráljuk az *alakzat* változót a létrehozott objektumra való hivatkozáshoz. A létrejövő alakzatokra a közös ős refenciájával hivatkozunk.

```

uses Geometria;
var
    alakzat:TPozicio=nil;

```

A választógomb-csoporton való kattintás törli az előző alakzatot és újat hoz létre.

```

procedure TForm1.rgAlakzatClick(Sender: TObject);
begin
    btnTorolClick(Sender);
    btnLetrehozClick(Sender);
    Beep;
end;

```

Az alakzat létrehozása a *btnLetrehozClick* eseménykezelőben, törlése a *btnTorolClick* eseménykezelőben történik.

```

// A választógommbal kijelölt alakzat létrehozása
procedure TForm1.btnLetrehozClick(Sender: TObject);
begin
    case rgAlakzat.ItemIndex of
        0 : alakzat:=TPont.Create(60,100, clYellow);
        1 : alakzat:=TKor.Create(60,100, clBlue,60);
        2 : alakzat:=TNegyzet.Create(60,100, clRed,120);
        3 : alakzat:=TTeglalap.Create(60,100, clGreen,120,180);
    end;
    btnLetrehoz.Enabled:=false;
    btnMozgat.Enabled:=true;
    btnTorol.Enabled:=true;
end;

```

```

// Ha az alakzat létezik, akkor töröljük azt
procedure TForm1.btnTorolClick(Sender: TObject);
begin
    if Assigned(Alakzat) then
        begin
            Alakzat.Rajzol(form1.color);
            Alakzat.Free;
            Alakzat:=nil;
        end;
    btnLetrehoz.Enabled:=True;
    btnMozgat.Enabled:=False;
    btnTorol.Enabled:=False;
end;

```

Mozgatáshoz a *Mozgat* metódust használhatjuk.

```

// Az alakzat mozgatása vízszintesen
procedure TForm1.btnMozgatClick(Sender: TObject);
var
    x : integer;
begin
    if Alakzat=nil then exit;
    x:=60;
    while x<panel1.left-60 do
        begin
            Alakzat.Mozgat(x,100);
            Sleep(60);
            inc(x,15);
        end;
    end;

```

A form létrehozásakor alapállapotot állítunk be, átméretezéskor újrarajzolunk.

```

// A form létrehozásakor beállítjuk a gombok léthatóságát
procedure TForm1.FormCreate(Sender: TObject);
begin
    btnLetrehoz.Enabled:=True;
    btnMozgat.Enabled:=False;
    btnTorol.Enabled:=False;
end;

// A form átméretezésekor is megjeleníti az alakzatot
procedure TForm1.FormResize(Sender: TObject);
begin
    form1.Repaint;
    if Assigned(alakzat) then Alakzat.Frissit;
end;

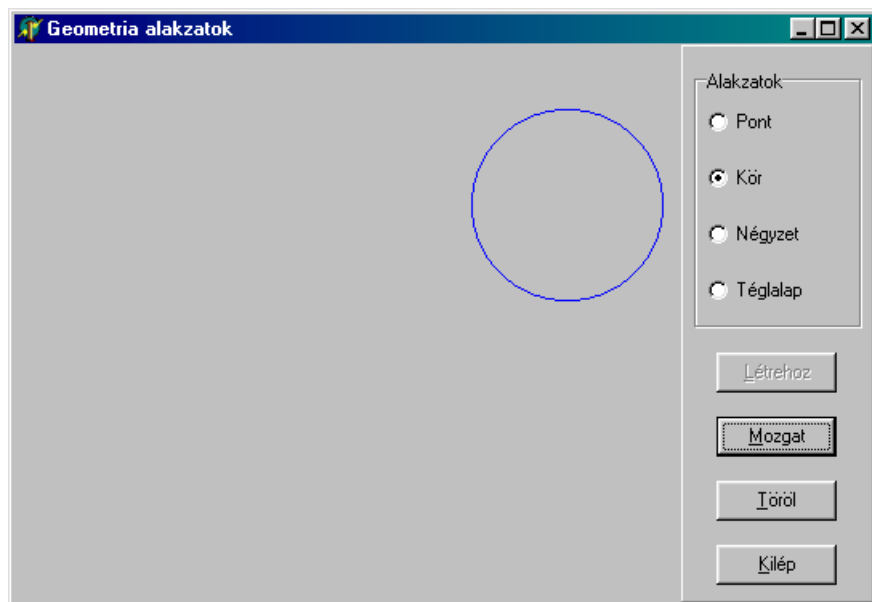
```

A kilépés gombbal lépünk ki.

```

// Kilépés az alkalmazásból
procedure TForm1.btnKilepClick(Sender: TObject);
begin
    btnTorolClick(Sender);
    form1.close;
end;

```

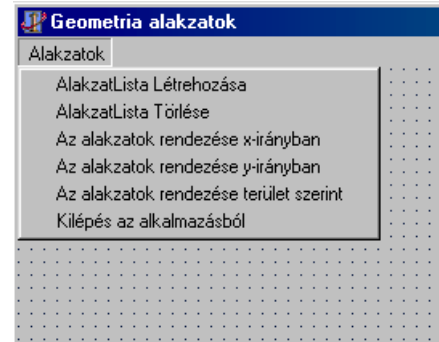




A *Geometria* feladatban definiált alakzatokkal töltünk fel egy *TObjectList* típusú objektumlistát! Az alkalmazás legyen képes megjeleníteni, különböző szempontok szerint rendezni és törölni a lista tartalmát! (*GrafGeomList*)

A programban menüvezérelten listába gyűjtött alakzatokat rajzolunk ki.

```
type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    Alakzatok1: TMenuItem;
    AListaLetrehozasa: TMenuItem;
    AListaTorlese: TMenuItem;
    Kilepes: TMenuItem;
    ListaRendezesX: TMenuItem;
    ListaRendezesY: TMenuItem;
    ListaRendezesTerulet: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure AListaLetrehozasaClick(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure AListaTorleseClick(Sender: TObject);
    procedure ListaRendezesXClick(Sender: TObject);
    procedure ListaRendezesYClick(Sender: TObject);
    procedure ListaRendezesTeruletClick(Sender: TObject);
    procedure KilepesClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```



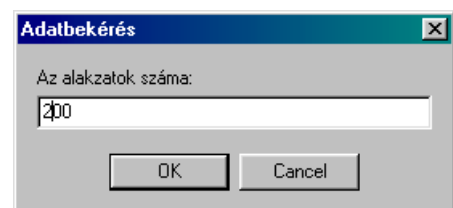
A programban a *GrafGeom* program *Geometria* modulját használjuk és az elemek tárolására az *OLista* objektumlistát deklaráljuk.

```
uses Geometria;
var
  OLista : TObjectList;
```

A form létrehozásakor létrehozuk az *OLista* listát, intézkedünk arról, hogy a lista elemei megszűnjenek, ha lekerülnek a listáról, továbbá beállítjuk a rajzolás vonalvastagságát.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  OLista:=TObjectList.Create;
  // A listából törölt objektum meg is szűnik
  OLista.OwnsObjects:=true;
  // 1 pixel vastag v
  Canvas.pen.width:=1;
  Randomize;
end;
```

Az „*AlakzatLista létrehozása*” menüpont választásakor bekérjük a lista hosszát, létrehozuk a listaelemeket, melyek színét és típusát a véletlenre bízuk. Az alakzatok a form területére kerülnek.



```

// A lista feltöltése alakzatokkal
procedure TForm1.AListaLetrehozasaClick(Sender: TObject);
var
    alakzat    : TPozicio;
    akod       : 0..3;
    i,x,y,a,b  : integer;
    szin       : TColor;
    s          : string;
    db         : integer;
begin
    // Ha nem adok meg értéket kilépünk
    s:='2000';
    if not InputQuery('Adatbekérés','Az alakzatok száma:',s) then exit;
    // A listában tárolt alakzatok megszüntetés
    OLista.Clear;
    form1.repaint;
    try
        db:=strtoint(s)
    except
        db:=1;
    end;
    for i:=1 to db do
        begin
            alakzat:=nil;
            akod:=random(4);
            szin:= RGB(random(256),random(256),random(256));
            // A form kliens területen belül marad az alakzat
            a:=20+random(150);
            b:=20+random(200);
            x:=1+a div 2+random(form1.clientwidth-a);
            y:=1+max(a,b) div 2+random(form1.clientheight-max(a,b));
            case akod of
                0 :   alakzat:=TPont.Create(x,y,szin);
                1 :   alakzat:=TKor.Create(x,y,szin,a div 2);
                2 :   alakzat:=TNegyzet.Create(x,y,szin,a);
                3 :   alakzat:=TTeglalap.Create(x,y,szin,a,b);
            end;
            OLista.Add(alakzat);
        end;
    end;

```

A listát törölhetjük az „AlakzatLista törlése” menüponttal.

```

// A lista elemeinek törlése
procedure TForm1.AListaTorleseClick(Sender: TObject);
begin
    OLista.Clear;
    form1.repaint;
end;

```

rendezhetjük az alakzatokat x-, y-koordinátáik szerint, illetve területük alapján. A rendezéshez a *Sort* metódust használjuk, úgy, hogy más-más összehasonlító függvényt adunk meg a sorrend meghatározásához.

```

// Az alakzatok rendezése az x-koordinátájuk alapján
procedure TForm1.ListaRendezesXClick(Sender: TObject);
    // A rendezéshez használt függvény
    function HasonlitX(Item1, Item2: Pointer): Integer;
    begin
        result:=round(TPozicio(Item1).GetX-TPozicio(Item2).GetX);
    end;
begin
    OLista.Sort(@HasonlitX);
    form1.repaint;
end;

```

```

// Az alakzatok rendezése az y-koordinátájuk alapján
procedure TForm1.ListaRendezesYClick(Sender: TObject);
  // A rendezéshez használt függvény
  function HasonlitY(Item1, Item2: Pointer): Integer;
  begin
    result:=round(TPozicio(Item1).GetY-TPozicio(Item2).GetY);
  end;
begin
  // Az elemek rendezése a típusuk alapján
  OLista.Sort(@HasonlitY);
  form1.repaint;
end;

// Az alakzatok rendezése a területük alapján
procedure TForm1.ListaRendezesTeruletClick(Sender: TObject);
  // A rendezéshez használt függvény
  function HasonlitTer(Item1, Item2: Pointer): Integer;
  begin
    result:=round(TPozicio(Item1).Terulet-TPozicio(Item2).Terulet);
  end;
begin
  // Az elemek rendezése a területük alapján
  OLista.Sort(@HasonlitTer);
  form1.repaint;
end;

```

Kirajzolás a *FormPaint* eseményben.

```

// A form újrafestétekor megjelenítjük az alakzatokat
procedure TForm1.FormPaint(Sender: TObject);
var
  i : integer;
begin
  with OLista do
    for i:=0 to Count-1 do
      if Assigned(Items[i]) then TPozicio(Items[i]).Frissit;
end;

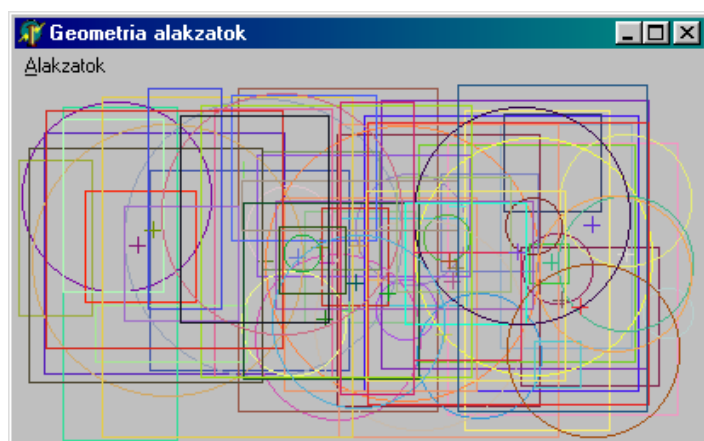
```

Kilépéskor takarítani kell.

```

// Kilépéskor töröljük a listaelemeket és
// a listaobjektumot
procedure TForm1.KilepesClick(Sender: TObject);
begin
  if Assigned(OLista) then
    begin
      OLista.Clear;
      OLista.Free;
    end;
  Form1.Close;
end;

```

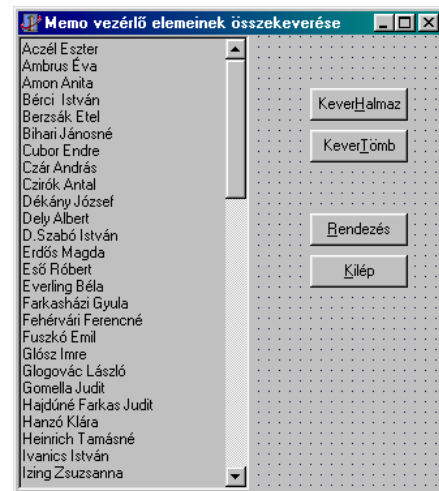




Írjunk programot, amely a *Memo* vezérlőben (lásd 3. fejezet) tárolt neveket összekeveri, illetve sorba rendezi! (*SKeveres*)

Helyezzük el a formon a *Memo1* szövegszerkesztőt, soraiban nevekkal! Tegyük két keverés gombot a formra. Az egyikkel a sorokat egy halmaz segítségével keverjük meg, a másik esetben pedig egy dinamikus tömbbel. A feltöltött elemeket a *Memo1.Sort* metódusával rendezzük sorba.

```
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    btmKeverHalmaz: TButton;
    btmKilep: TButton;
    btnRendezes: TButton;
    btnKeverTomb: TButton;
    procedure btmKeverHalmazClick(Sender: TObject);
    procedure btnRendezesClick(Sender: TObject);
    procedure btnKeverTombClick(Sender: TObject);
    procedure btmKilepClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```



Ha 256 elemnél kevesebb nevünk van., akkor indexként használhatunk halmazt is, egyébként csak a dinamikus tömböt (*Jelzo*). Az index elemeket összekeverjük és a *Memo1.Lines* sorait megfelelően cseréljük.

```
// A Memo1 elemeinek keverése halmaz segítségével
procedure TForm1.btmKeverHalmazClick(Sender: TObject);
var
  jelzo : set of byte;
  i, a, b, db : integer;
begin
  db:=Memo1.Lines.count;
  if db>256 then
    begin
      ShowMessage('Túl sok elem, halmazzal nem keverhető össze.');
```

```
      exit;
    end;
  jelzo:=[];
  // A jelzőhalmaz feltöltése a lehetséges indexekkel
  for i:=0 to db-1 do
    include(Jelzo,i);
  // Mindegyik elemet felcserélünk egy
  // véletlenpozíción elhelyezkedővel
  for a:=0 to db-1 do
    begin
      repeat
        b:=Random(db);
      until (b in jelzo) or (jelzo=[]);
      exclude(jelzo,b);
      Memo1.Lines.Exchange(a,b);
    end;
  end;
end;

// A Memo1 elemeinek keverése dinamikus tömb segítségével
procedure TForm1.btnKeverTombClick(Sender: TObject);
var
  jelzo : array of boolean;
  i, a, b, db, cnt : integer;
begin
  db:=Memo1.Lines.count;
  SetLength(jelzo,db);
```

```

// A jelzőtömb feltöltése true értékkel
for i:=0 to db-1 do
  Jelzo[i]:=true;
// Mindegyik elemet felcserélünk egy
// véletlenpozíció elhelyezkedővel
cnt:=0;
for a:=0 to db-1 do
  begin
    repeat
      b:=Random(db);
    until Jelzo[b] or (cnt=db);
    Jelzo[b]:=false;
    inc(cnt);
    Memo1.Lines.Exchange(a,b);
  end;
end;

```

A neveket áttöltjük a **TStringList** típusú *sl*-be és a **Sort** metódussal sorba rendezzük, majd visszatöltjük

```

// A sorok rendezesehez TStringList objektumot használunk
procedure TForm1.btnRendezesClick(Sender: TObject);
var
  sl : TStringList;
begin
  sl:=TStringList.Create;
  sl.Text:=Memo1.Lines.Text;
  sl.Sort;
  Memo1.Lines.Text:=sl.Text;
  sl.free;
end;

```

