

F0. Kivételek kezelése

1. Egyszerű számítógép konzolalkalmazás, hibakezeléssel [Szgep1](#)
2. Egyszerű számítógép alkalmazás, hibakezeléssel [Szgep2](#)
3. Könyvtárak olvasása kivételkezeléssel [DirLista](#)
4. Állomány másolása hibakezeléssel [FMasolo](#)



Készítsünk fordított lengyel logikát használó egyszerű számológép konzol-alkalmazást, melynek működése során fellépő hibákat kivételek segítségével kezeljük! (*Szgepl*)

Használunk kell a *Math* modult is.

```
uses SysUtils, Math;
```

A fordított lengyel logika szerint először a műveleti jelet adjuk meg, aztán az operandusokat. A lehetséges műveleti jeleket az *mjelek*, a kilépést kérő jeleket a *kjelek* karakterhalmaz tartalmazza.

```
const
  mjelek = ['+', '-', '*', '/', '^'];
  kjelek = ['q', 'x', 'Q', 'X'];
```

A rossz műveleti jelet saját kivételosztállyal kezeljük

```
// Saját kivételosztály létrehozása
EMJelHiba = class(ERangeError);
```

64-bites egész számokat használunk, *a* és *b* az operátorok, *c* az eredmény, *op* az operátor és a *vege* jelöli a felhasználó kilépési szándékát.

```
type
  integer = int64;
var
  a, b : Integer; // szám1, szám2
  c : Integer; // az eredmény
  op : char; // az operátor
  vege : boolean;
```

Az adatokat a *Beolvas* eljárás szedi össze. Az adatbevitel során először az operátort kell megadni, saját kivétellel kezeljük a rossz operátort.

```
procedure Beolvas;
begin
  write(':');
  read(op);
  // Rossz operátor esetén saját kivétel kiváltása
  if not (op in (mjelek + kjelek)) then
    begin
      readln;
      raise EMJelHiba.Create('A műveleti jel csak +,-,*,/,^ lehet!');
    end;
  { Program befejezését jelző operátor esetében az EAbort kivételt
    kiváltó Abort függvény hívása }
  if op in kjelek then Abort;
  readln(a, b);
end;
```

A számításokat az operátornak megfelelően a *Szamol* függvény végzi:

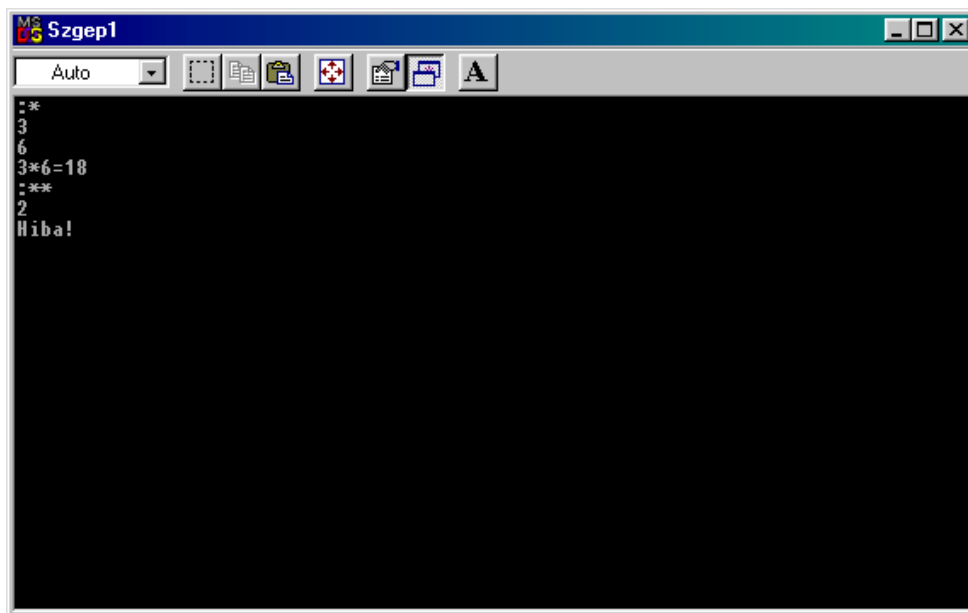
```
function Szamol: Integer;
begin
  result:=0;
  case op of
    '+': result:=a+b; // összeadás
    '-': result:=a-b; // kivonás
    '*': result:=a*b; // szorzás
    '/': result:=a div b; // osztás
    '^': result:=trunc(power(a,b)); // hatványozás
  end;
end;
```

A *Kiir* írja ki az eredményeket

```
procedure Kiir;  
begin  
    writeln(a,op,b,'=',c);  
end;
```

A főprogramban hívjuk a fenti alprogramokat és kezeljük a számítási hibákat.

```
// A főprogram  
begin  
    vege := False;  
    repeat  
        try  
            Beolvas;  
            c:=Szamol;  
            Kiir;  
        except  
            // Ötféle kivétel kezelése  
            on E: EMJelHiba do writeln(E.Message);  
            on EDivByZero do writeln('Hiba: nullaval osztas!');  
            on EIntOverflow do writeln('Hiba: tulcsordulas!');  
            on EInvalidOp do writeln('Matematikai hiba!');  
            on EAbort do vege := True;  
        else  
            // Minden más kivétel esetén  
            writeln('Hiba!');  
            readln;  
        end;  
    until vege;  
end.
```

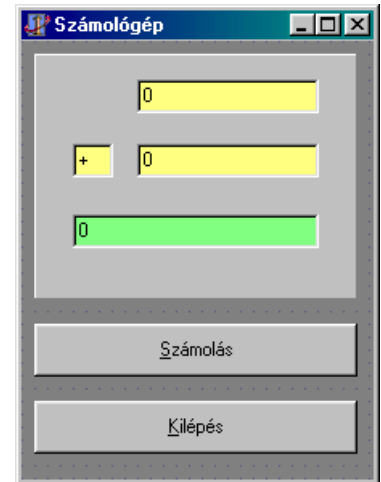




A 3. fejezet áttanulmányozása után, készítsük el az *Szgep1* program grafikus felülettel rendelkező változatát! (*Szgep2*)

A programban grafikus felhasználói felületre cseréljük az *Szgep1* program adatmegadási lehetőségeit. Tervezzük meg a felhasználói felületet. Az *Edit1* és *Edit3* vezérlők az operandusok-, az *Edit2* az operátor megadását szolgálja. Az *Edit4* jeleníti meg az eredményt. A *Számolás* (*Button1*) gomb indítja a számítást

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    Panel1: TPanel;
    Edit1: TEdit;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit2: TEdit;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Edit1KeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure Edit3KeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure Edit2KeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure Edit1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```



A deklarációk hasonlóak az *Szgep1* programéhoz.

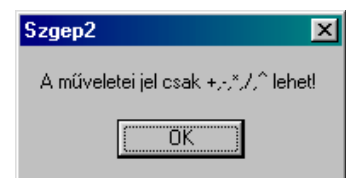
```
type
  // Saját kivételosztály létrehozása
  EMJelHiba = class(ERangeError);
var
  a, b : Integer; // szám1, szám2
  c : Integer; // az eredmény
  op : char; // az operátor
```

Az adatbevitelkor sztringeket kell számmá konvertálni.

```
// Adatbevitel
procedure Beolvas;
begin
  op:=Form1.edit2.text[1];
  a:=strtoint(Form1.Edit1.Text);
  b:=strtoint(Form1.Edit3.Text);
end;
```

A számítás ugyanaz, mint az *Szgep1* esetén.

```
// A számolást végző függvény
function Szamol: Integer;
begin
  case op of
    '+': result:=a+b; // összeadás
    '-': result:=a-b; // kivonás
    '*': result:=a*b; // szorzás
    '/': result:=a div b; // osztás
    '^': result:=trunc(power(a,b)); // hatványozás
  else
    // Rossz operator esetében saját kivétel kiváltása
    raise EMJelHiba.Create('A műveleti jel csak +,-,*,/,^ lehet!');
  end;
end;
```



A kiírás az *Edit4* segítségével történik.

```
procedure Kiir;  
begin  
    Form1.Edit4.Text:=Inttostr(c);  
end;
```

Az [Szgepl](#) főprogramját a *Calculate* eljárás helyettesíti.

```
procedure Calculate;  
begin  
    try  
        Beolvas;  
        c:=Szamol;  
        Kiir;  
    except  
        // Kivételkezelők  
        on E: EMJelHiba do ShowMessage(E.Message);  
        on EDivByZero do ShowMessage('Nullával osztás!');  
        on EZeroDivide do ShowMessage('Nullával osztás!');  
        on EIntOverflow do ShowMessage('Túlcsordulás!');  
        on EInvalidOp do ShowMessage('Matematikai hiba!');  
        on EConvertError do ShowMessage('Hibás adatbevitel!');  
    else  
        // Minden más kivétel esetén  
        ShowMessage('Hiba történt!');  
    end;  
end;
```

A *Button1* (Számolás) gomb megnyomásakor indítjuk a számítást.

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    Calculate;  
    Edit1.Setfocus;  
end;
```

A program indításakor kezdőértékeket adunk.

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    Edit1.Text:='0';  
    Edit2.Text:='+';  
    Edit3.Text:='0';  
    Edit4.Text:='0';  
end;
```

A kilépés is gombnyomásra történik.

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

Az adatbeviteli ablakokban figyelni kell az *<Enter>* billentyűt is.

```
procedure TForm1.Edit1KeyDown(Sender: TObject; var Key: Word;  
    Shift: TShiftState);  
begin  
    if key=13 then  
        begin  
            Edit3.SetFocus;  
            Key:=0;  
        end;  
    end;  
end;
```

```

procedure TForm1.Edit3KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if key=13 then
    begin
      Edit2.SetFocus;
      Key:=0;
    end;
end;

```

```

procedure TForm1.Edit2KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if key=13 then
    begin
      Button1Click(Sender);
      Key:=0;
    end;
end;

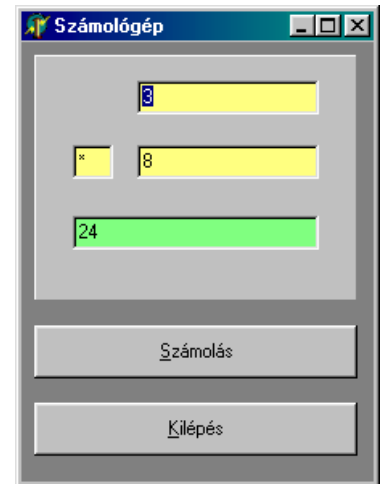
```

Az eredmény érvénytelen, ha az adatok változnak.

```

procedure TForm1.Edit1Change(Sender: TObject);
begin
  Edit4.Clear;
end;

```





Készítsünk alkalmazást, amely felolvassa az aktuális mappa tartalmát és sztringlistába tárolja a bejegyzéseket! A műveletek biztonságáról a kivételkezelés eszközeivel gondoskodjunk! (*DirLista*)

A *GetFileList* függvény olvassa a paramétereként megadott könyvtár tartalmát és eredményként *TStringList*-ben tárolja az adatokat. Ha az olvasás sikertelen, akkor felszabadítjuk a listát. Az állományok kezelésére a *FindFirst* és *FindNext* függvényeket használjuk. A *FindClose* függvényt minden esetben meghívjuk (Lásd a 7. fejezetet).

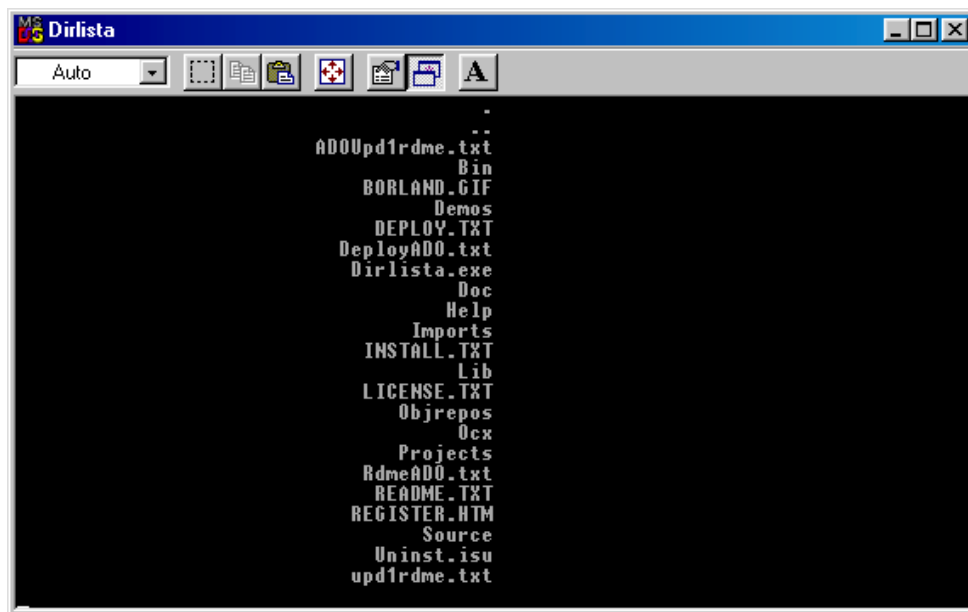
```
{ $Apptype Console }
program DirLista;

uses Classes, SysUtils;

// A felolvasást végző eljárás
function GetFileList(const Path: string): TStringList;
var
  hiba : Integer;
  sr   : TSearchRec;
begin
  Result:=TStringList.Create;           // a lista létrehozása
  try
    try
      // a keresés inicializálása
      hiba:=FindFirst(Path, faAnyFile, sr);
      while hiba=0 do
        begin
          Result.Add(sr.Name);
          hiba:=FindNext(sr);
        end;
      finally
        // A findclose hívás minden esetben
        // végrehajtódik
        FindClose(sr);
      end;
    except
      // Kivétel fellépte esetén:
      Result.Free; // a lista felszabadítása
      raise;       // az eredeti kivétel újraaktivizálása
    end;
  end;
end;
```

A főprogram az aktuális könyvtárra meghívja a *GetFileList* függvényt, és az eredményt sorba rendezi

```
var
  DirTart : TStringList;
  i       : integer;
  path    : string;
begin
  GetDir(0, path);
  DirTart:=GetFileList(path+'\\*.*');
  DirTart.Sort;
  for i:=0 to DirTart.Count-1 do
    writeln(Format('%40s', [DirTart.Strings[i]]));
    readln;
  end.
```





Írjunk állománymásoló programot. A fájl biztonságos másolását a **try..finally** kivételkezelő többszintű alkalmazásával valósítsuk meg! (*FMasolo*)

A másolást a *Masol* eljárás végzi, mely adatfolyamként nyitja - a paraméterként megadott - forrás (*MitNev*) és a cél (*Hovanev*) állományt és az egyiket a másikba másolja (lásd 7.2. fejezet). Az eljárás futása során hibakezeléssel figyeljük az állományok megnyitásának és adatfolyamba való másolásának sikerét.

A másolást végző eljárás, mindkét állományt adatfolyamként (stream) érjük el:

```
procedure Masol( const MitNev, HovaNev: string );
var
  ff, tf: TFileStream;
begin
  // A forrásállomány megnyitása
  ff := TFileStream.Create( MitNev, fmOpenRead );
  try
    // A célállomány megnyitása
    tf := TFileStream.Create( HovaNev, fmOpenWrite or fmCreate );
    try
      // Másolás
      tf.CopyFrom(ff, ff.Size);
    finally
      // A célállomány lezárása
      tf.Free;
    end;
  finally
    // A forrásállomány lezárása
    ff.Free;
  end;
end;
```

A *Masol* függvényt hívó főprogram, amely a *c:\Autoexec.bat* állományt a *c:\Autoexec.\$\$\$* fájlba másolja:

```
begin
  {A másolás vezérlése hibafigyeléssel}
  try
    Masol('c:\autoexec.bat', 'c:\autoexec.$$$');
    writeln('A masolas sikeres volt!');
  except
    writeln('Sikertelen volt a masolas!');
  end;
  readln;
end.
```