Visual Basic 2005 Express Edition

Kiegészítések az Informatika 10. tankönyv Algoritmusok és adatok fejezetéhez

(Nemzeti Tankönyvkiadó, 2006, r.sz.: 16272)

Összeállította: Juhász Tibor (2008)

NEM LEKTORÁLT VÁLTOZAT

A dokumentumban előforduló hibákat és egyéb megjegyzéseket kérjük a juhaszt@zmgzeg.sulinet.hu címen jelezni.

A 10. osztályos informatika-tankönyv Pascal és Visual Basic Script nyelveken mutatja be az algoritmusoknak a tanterv által előírt számítógépes megvalósítását. Bár erre a célra a VBScriptet tartjuk a legalkalmasabbnak, az informatika érettségin és OKTV-n használható szoftverek miatt kiegészítjük a tankönyvet aVisual Basic programozási nyelvre vonatkozó tudnivalókkal.

A *Visual Basic 2005 Express Edition* ingyenesen letölthető a Microsoft webhelyéről, és korlátozások nélkül felhasználható. A letöltési-telepítési útmutatót külön fájlban részletezzük.

A Microsoft webhelyén a *Visual Studio 2008 Express Edition* részeként megtalálható a *Visual Basic 2008 Express Edition*, melynek telepítése, használata nagyon hasonlít a 2005-ös verzióhoz.

Az alábbiakban a következő elnevezéseket alkalmazzuk. A tankönyv fejezetekből áll. Egy fejezet például: *Algoritmusok és adatok* (99. oldal). A fejezetet leckék alkotják. Az algoritmusok fejezet második leckéje például: *Algoritmusleíró eszközök* (102. oldal). A leckéket alcímek tagolják kisebb részekre. Az *Algoritmusleíró eszközök* lecke első alcíme például: *Szemléltetés folyamatábrával*.

A tankönyvben nem szereplő leckéket, alcímeket az *Új dokumentum* ikonjával jelöljük:  . A tankönyvben szereplő alcímek kiegészítésére az előbbi ikonban látható plusz-jel utal: 

A Visual Basic szintaxisa, utasításai nagyon hasonlítanak a tankönyvben bemutatott Visual Basic Scripthez. Ha külön nem említjük, akkor a tankönyv Visual Basic Scriptre vonatkozó megállapításai érvényesek a Visual Basicre is.

A kiegészítésben a Visual Basic sajátosságaira a program logójával utalunk:  . Az algoritmusok, példák jelölésére megtartjuk a tankönyvben alkalmazott ikonokat.

A kiegészítéshez mellékeljük a tankönyvben szereplő mintapéldák Visual Basic forráskódját (a projektfájlokat), az integrált fejlesztői környezet ismertetését (*Visual Basic – első lépések*) és a *Visual Basic 2005 Express Edition* telepítési útmutatóját. Mivel a Visual Basic projektek több fájlból és mappából állnak, a forráskódok mellett megadjuk a megfelelő projekt mappanevét.

ALGORITMUSOK ÉS ADATOK

(Bevezetés, 99. oldal)

A *Visual Basic Express Edition* szabadon letölthető a Microsoft webhelyéről:

www.microsoft.com/express/2005

A telepítés elvégezhető az Internetről, illetve a letöltött képfájl (*.img* vagy *.iso*) CD-re írásával, majd a telepítőprogram indításával. Ha a telepítést az Internetről végezzük, akkor 30 napon belül regisztrálni kell a terméket. A regisztráció ingyenes. CD-ről történő telepítés esetén nincs szükség regisztrációra. A telepítés részleteit lásd a Telepítési útmutatóban.

Al-Hvárizmi öröksége (100. oldal)

Eljárások

Az algoritmusokban gyakran előfordul olyan utasításcsoport, amely több helyen is megismétlődik. Ha ezt az utasításcsoportot névvel látjuk el, akkor az ismételt beírás helyett elegendő az adott névvel hivatkozni rá. Ahol ezt a nevet látjuk az algoritmusban, ott végre kell hajtani az utasításcsoport utasításait.

Eljárás (szubrutin): névvel ellátott, elkülönített utasításcsoport, amely általában önálló részfeladatot old meg.

Ha például egy nagyobb algoritmusban többször is előfordul az SMS-küldés, akkor foglaljuk az utasításokat eljárásba:

```
Eljárás SMS-küldés
    Lépj be az Üzenetek mappába.
    Válaszd ki az Új üzenet küldése parancsot.
    Válaszd ki a címzettet a Telefonkönyv listájából.
    Írd meg az üzenetet.
    Küldd el az üzenetet.
Eljárás vége
```

Az utasítások végrehajtásához a nagyobb algoritmus megfelelő helyére beírjuk az eljárás nevét:

```
… (utasítások)
SMS-küldés
… (utasítások)
```

Amikor az algoritmusban elérkezünk az eljárás nevéhez, akkor az SMS-küldés utasításaival folytatjuk a feldolgozást. Ezt a folyamatot **az eljárás meghívásának** nevezzük. Az eljárás végén visszatérünk a hívást követő utasításhoz.

Előbbi megoldásunk hiányossága, hogy az eljárás végrehajtásakor nem tudjuk, kinek és milyen tartalommal küldjünk üzenetet. Az eljárásnak átadhatjuk ezeket az adatokat. Ha egy eljárás ilyen értékeket vár a meghívása során, akkor a definícióban az eljárás neve után zárójelbe téve felsoroljuk az adatok megnevezéseit, melyeket **az eljárás paramétereinek** hívunk:

```
Eljárás SMS-küldés(Címzett, Üzenet)
```

Az eljárás utasításaiban felhasználhatjuk a paramétereket:

Eljárás SMS-küldés(Címzett, Üzenet)
    Lépj be az Üzenetek mappába.
    Válaszd ki az Új üzenet küldése parancsot.
    Válaszd ki a *Címzett*-et a Telefonkönyv listájából.
    Írd be az *Üzenet*-et.
    Küldd el az üzenetet.
Eljárás vége

Az eljárás meghívásánál a paraméterek helyére azok konkrét értékét írjuk be, például:

…
SMS-küldés(Laci, ”Holnap délután érkezem.”)
…

Az eljárás végrehajtásánál a paraméterek helyére azok konkrét értéke kerül:

Lépj be az Üzenetek mappába.
Válaszd ki az Új üzenet küldése parancsot.
Válaszd ki a Lacit a Telefonkönyv listájából.
Írd be a ”Holnap délután érkezem.” üzenetet.
Küldd el az üzenetet.

A paraméter konkrét értékét gyakran **argumentumnak** nevezik.

Az eljárások nem csak a több helyen is meghívásra kerülő utasítássorozatokat foghatják össze. Eljárások szervezése növeli az algoritmus áttekinthetőségét. Segítségükkel elkülöníthetjük a részfeladatokat, a nagy, összetett algoritmust kisebb, többé-kevésbé önálló részekre oszthatjuk fel.

Algoritmusok és a számítógép (104. oldal)

Köztes nyelv és felügyelt kód
(kiegészítő anyag)

A modern programozási nyelveknél merült fel az az igény, hogy a forráskód ne kötődjön egy meghatározott operációs rendszerhez, illetve egy összetett feladat egyes részeit egymástól eltérő programozási nyelveken készíthessék el a programozók. Ezért a Visual Basicben megírt forráskódot a fordítóprogram a gépi kód helyett egy közbenső nyelvre, az úgynevezett MIL-re fordítja le (MIL: Microsoft Intermediate Language, Microsoft köztes nyelv). A közbenső vagy **köztes nyelv** már független a forráskódban használt programozási nyelvtől, továbbá független az operációs rendszertől és a hardverkörnyezettől is.

A MIL-változatból a program futtatásakor a **futás előtti fordító** hozza létre a mikroprocesszor számára is érhető gépi kódot (JIT: Just In Time, futás előtt). Bár a JIT-fordító a futás során lép működésbe, de a köztes nyelvről történő fordítás sokkal gyorsabb, mint az eredeti forráskód fordítása gépi kódra.

A JIT-et a Windows bővítése, az úgynevezett **.NET** (ejtsd: dotnet) tartalmazza (.NET Framework, .NET keretrendszer). A .NET része továbbá a **közös nyelvi futtatómodu**l, amely a futás előtti fordításon kívül a programok futtatásáért, az erőforrások elosztásáért felelős (CLR: Common Language Runtime, közös nyelvi futtató). A CLR alkalmazkodik a számítógép hardver- és szoftverelemeihez, ezekkel nem kell a programozónak törődnie. Mivel a CLR felügyeli a program végrehajtását és az erőforrások biztonságos felhasználását, a CLR által futtatott kódot **felügyelt kódnak** nevezik.

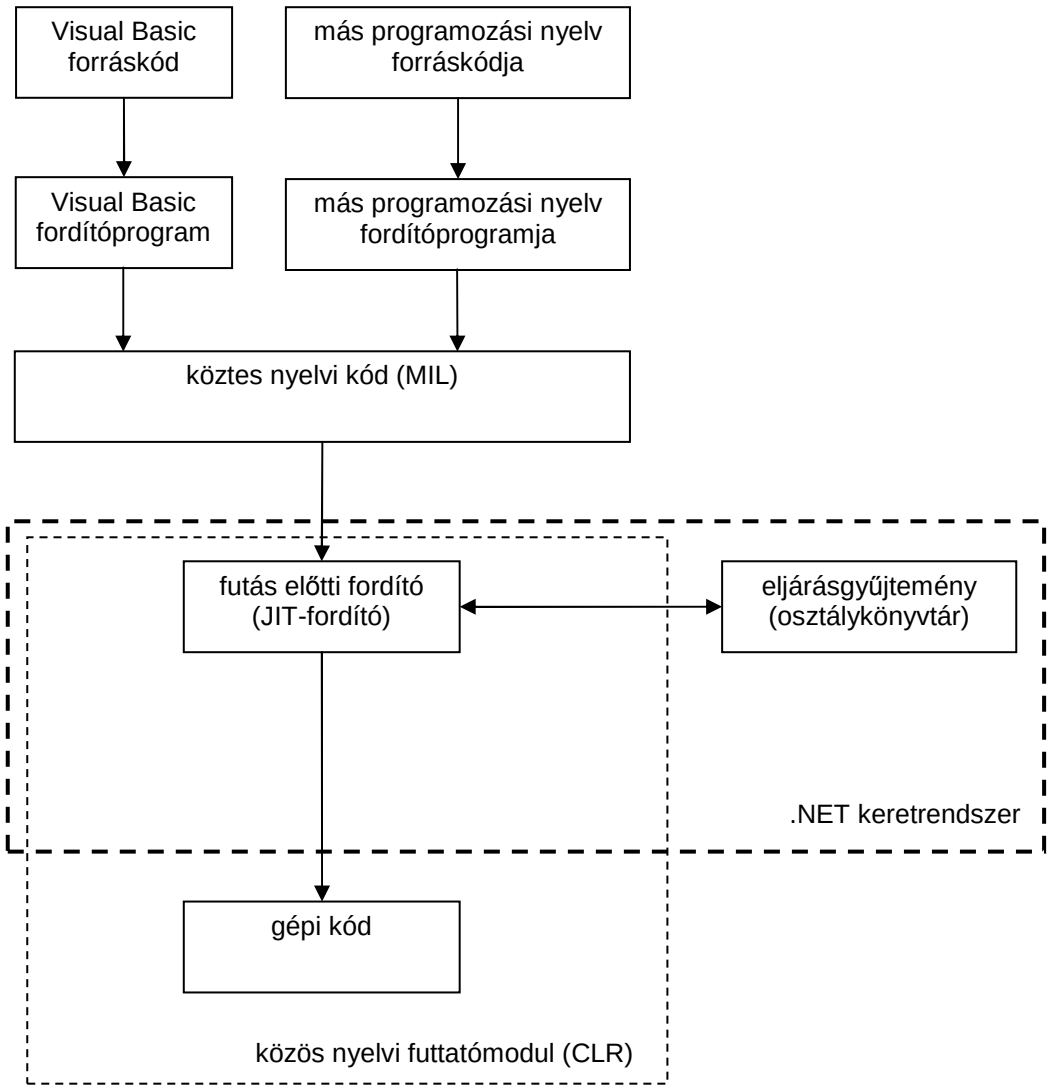
A .NET-ben találunk még egy több ezer eljárásból álló gyűjteményt[1]. Ezeket az eljárásokat felhasználhatjuk programjainkban. Ezzel rengeteg részfeladat programozásától kímélhetjük meg magunkat.

A .NET Framework már beépült a Windows Vista operációs rendszerbe. Windows XP használata esetén a .NET keretrendszer letölthető a Microsoft webhelyéről. Telepítését elvégzik azok a

[1] Pontosabban szólva osztálykönyvtárat.

programok is, melyeknek szüksége van rá. A Visual Studio (Visual Basic) 2005 Express Edition például szükség esetén telepíti a .NET Framework 2.0-ás változatát.

Megjegyezzük, hogy az első, platformfüggetlenségre törekvő nyelv, a Java az 1990-es évek közepén jött létre. A Java fordítóprogramja által előállított köztes kódot bájtkódnak nevezik. Ezt a Java futtatómodulja hajtja végre (Java virtuális gép).



A .NET keretrendszer szerkezete és a fordítás folyamata

A programozási nyelvek elemei (107. oldal)

Programozás Visual Basicben

Az alábbiakban csak a tankönyv szerkezetének megfelelő kiegészítés következik. Az integrált fejlesztői környezet kezelését, a programok megírását, futtatását, a hibajavítást részletesebben lásd: ***Visual Basic 2005 – első lépések****.*

A Visual Basic programozási nyelvet a Microsoft fejlesztette ki 1991-ben. Bár a Kemény János és Thomas Kurtz által 1964-ben megalkotott BASIC[2] nyelven alapul, jelenleg a Visual Basic egy modern, objektum-orientált, grafikus fejlesztői környezettel ellátott negyedik generációs programozási nyelv. Legújabb kiadása, a *Visual Basic 2005* (illetve 2008) több más programozási nyelv fejlesztői környezetével együtt a *Microsoft Visual Studio* részét képezi. *Express Edition* változatuk ingyenesen letölthető a Microsoft webhelyéről, és korlátozás nélkül felhasználható:

http://www.microsoft.com/express/2005

A Visual Basic a Windows alkalmazások létrehozásán túl kiválóan alkalmas adatbázisok kezelésére, webszerverek programozására, Office-alkalmazások vezérlésére, tenyérgépek (PDA) vagy okostelefonok programjainak megírására, hálózati kommunikáció kialakítására. Az Office-alkalmazások tartalmazzák a nyelv egy változatát (VBA: Visual Basic for Applications), mellyel makrókat készíthetünk.

A *Visual Basic 2005 Express Edition* fejlesztői környezete Windows programok és konzolalkalmazások készítését teszi lehetővé. A konzolalkalmazások nem használják ki a Windows grafikus felhasználói felületét. A felhasználóval az úgynevezett parancssori ablakon keresztül tartják a kapcsolatot. Első programunkat konzolalkalmazásként készítjük el.

Alkalmazásunkat egy Visual Basic projekt tartalmazza. A projekt modulokból áll. Egy modult a *Module … End Module* utasítások között helyezkedik el. A modul eljárásokat tartalmaz. Egy eljárást a *Sub … End Sub* utasítások határolnak. Mind a modul, mind az eljárás első sorában megadjuk a modul/eljárás nevét. A Visual Basic egy új projekt létrehozásánál előre elkészíti a modul és egy *Main* nevű eljárás keretét. A konzolalkalmazásoknál kötelező létrehozni a *Main* eljárást. A program futtatásánál ennek az eljárásnak az utasításai kerülnek végrehajtásra.

```
' Első Visual Basic programunk (Első mappa)
Module Module1
  ' A program fő-eljárása
  Sub Main()
    Console.WriteLine("Ez az első program.") ' sor kiírása
    Console.WriteLine("Ez a második sor szövege.")
    Console.ReadLine()
  End Sub
End Module
```

Első Visual Basic programunk

A *Console.WriteLine* utasítás az utána lévő zárójelek közé írt értéket írja ki a képernyőre. A kiírás végén sort emel. Ha karaktersorozatot akarunk megjeleníteni, akkor azt idézőjelek közé kell zárni. Az eljárás végén álló *Console.ReadLine()* hatására a program futása csak egy Enter lenyomására fejeződik be. Nélküle egy pillanat alatt eltűnne az ablak a képernyőről anélkül, hogy el tudnánk olvasni a benne lévő szöveget.

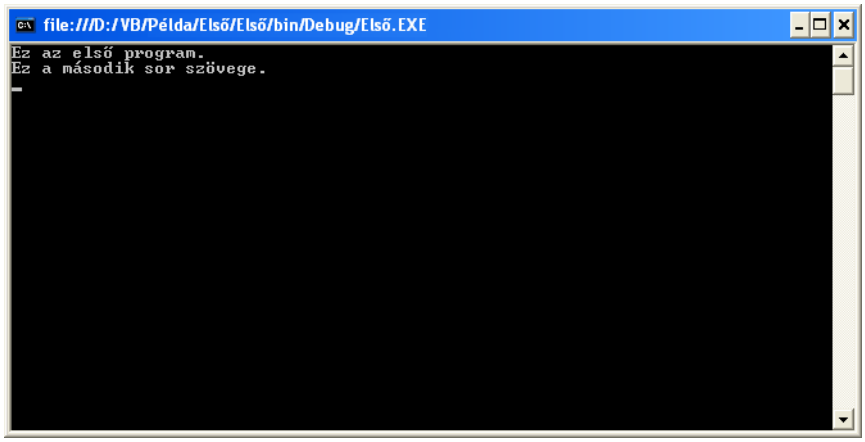
A forráskódban egy aposzrófjelet követően megjegyzés állhat. A megjegyzések megkönnyítik a program értelmezését, utólagos javítását, módosítását.

A kódszerkesztő ablakban egészítsük ki a *Main* eljárást programunk utasításaival! Minden utasítás külön sorban álljon! Ha túl hosszú lenne a sor, akkor egy szóköz és egy aláhúzásjel (_) beírása után a következő sorban folytathatjuk az utasítást.

[2] BASIC: Beginner's All-Purpose Symbolic Instruction Code, kezdők általános célú, szimbolikus utasításkódja.

A fordítóprogram nem különbözteti meg egymástól a kis- és nagybetűket, de a kódszerkesztő a kulcsszavak kezdőbetűit átírja nagybetűkre.

Futtassuk a programot, majd zárjuk be az ablakot!



Az első program

Elemi adattípusok (110. oldal)

Változók a programokban

Visual Basicben a változónevek tartalmazhatnak ékezetes karaktereket is. Ezzel olvashatóbbá, áttekinthetőbbé válik a program forráskódja. Például:

```
Név, Város, Telefonszám
```

A változók típusai

A Visual Basic legfontosabb típusai jelölésükben megfelelnek a Visual Basic Script típusainak (lásd: 111. oldal táblázata a tankönyvben). Az *integer* típus értéke azonban –2,1 milliárdtól +2,1 milliárdig, a *long* típus pedig $–9·10^{18}$-tól $+9·10^{18}$-ig változhat.[3]

A forráskódba a tizedesvessző helyett tizedespontot írjunk!

A változók deklarálása

A deklarálást a *Dim* kulcsszóval végezzük. Utána megadjuk az egyes változók nevét, majd az *As* kulcsszó után a típust. Új sor esetén meg kell ismételni a *Dim* kulcsszót:

```
Dim Név As String
Dim Szül_év, Szül_hónap As Integer, Kamatláb As Single
```

A forráskód beírásakor az *As* után némi szünettel megjelenik az intelligens segítség a típusok felsorolásával.

A változóknak deklaráláskor kezdőértéket adhatunk:

```
Dim Név As String = "Laci"
```

A változók deklarálásánál célszerű nagy kezdőbetűvel írni a változónevet. Ekkor a forráskód begépelésénél a kisbetűvel írt, felismert neveket a kódszerkesztő automatikusan átírja nagybetűsre. Ezzel megkönnyíti a hibakeresést.

A *Tools*/*Options* menüparancs *Projects and Solutions/VB Defaults* csoportjában az *Option Explicit* tulajdonság *Off*-ra állításával kikapcsolhatjuk a kötelező változódeklarálást, de ezt nem ajánljuk. Az *Option Strict* értékénél viszont hagyjuk meg az *Off* beállítást. Ezzel megkönnyítjük az automatikus típuskonverziót (lásd később).

[3] A VBScript *integer* típusának a Visual Basic *short* típusa felel meg.

Konstansokat a Visual Basic Script szintaxisának megfelelően deklarálhatunk:

```
Const Pi = 3.14159265
```

Megadhatjuk azonban a konstans típusát:

```
Const Pi As Single = 3.14159265
```

Ez utóbbi esetben a Visual Basic a megadott értéket 6 tizedesjegyre kerekíti.

Értékadás a változóknak (113. oldal)

Értékadó utasítások

Az értékadó utasítások szintaxisa, a műveletek jelölése (beleértve a hatványozás és maradékos osztás jelét) pontosan megfelel a Visual Basic Scriptnek. A kódszerkesztő az áttekinthetőség növelésére szóközöket illeszthet be a kifejezésekbe.

A változó értékének módosítását egyszerűsíti az úgynevezett **értékmódosító utasítás**, amelyben az értékadás egyenlőségjele elé egy műveleti jelet írunk. Például:

```
Alap += Kamat
```

Ez egyenértékű a következő utasítással:

```
Alap = Alap + Kamat
```

A Windows-alkalmazások készítésénél különösen hasznos lesz a számunkra a karakterláncok bővítése értékmódosító utasítással:

```
Név = Vezetéknév
Név &= " " & Keresztnév
```

A változók értékének megjelenítése

A változók értékét a *Console.WriteLine* utasítással írhatjuk ki a képernyőre. A *WriteLine* után zárójelben változónév vagy kifejezés állhat, beleértve egyetlen számot, illetve karakterláncot:

```
Console.WriteLine(Alap)
Console.WriteLine(Alap + Kamat)
```

A *WriteLine* után a képernyőn soremelés következik.

Ha több értéket szeretnénk megjeleníteni, akkor azokat az & jellel fűzzük össze[4]:

```
Console.WriteLine("Összesen: " & Alap + Kamat)
```

Tizedestörtek kiírásánál a területi beállításoknak megfelelő elválasztójel jelenik meg (például tizedesvessző). Figyeljük meg, hogy a *WriteLine* paramétereként összefűzhetjük a szövegeket, számokat és a kifejezések értékét is.

A **bank** program kiszámítja a bankban lévő pénzünk értékét *HónapokSzáma* hónap múlva.

A kiírás során a Visual Basic Scripthez hasonlóan a *FormatNumber* függvénnyel adhatjuk meg a tizedesjegyek számát:

```
FormatNumber(változónév, tizedesjegyek_száma)
```

Például: `FormatNumber(Alap, 1)`

[4] A műveletek közül az & kerül utoljára végrehajtásra.

Adatok beolvasása

Az adatokat a *Console.ReadLine()* utasítással olvassuk be. A *ReadLine* megvárja egy karaktersorozat begépelését, majd az Enter lenyomásával folytatódik a program futása. A beírt sztringet értékadó utasításban használhatjuk fel:

```
változónév = Console.ReadLine()
```

Célszerű a *ReadLine* előtt kiírni, hogy mit is vár tőlünk a program:

```
Console.WriteLine("Írd be a neved!")
Név = Console.ReadLine()
```

Ha a *WriteLine* helyett a *Write* utasítást alkalmazzuk, akkor a program nem emel sort a kiírás után:

```
Console.Write("Név: ")
Név = Console.ReadLine()
```

Vegyük figyelembe, hogy a *ReadLine* mindig szövegként kezeli a begépelt karaktereket. Ha numerikus változónak adunk értéket, akkor a sztringet számmá kell átalakítanunk. Az átalakításhoz használható függvények megegyeznek a Visual Basic Script függvényeivel (lásd a tankönyv 115. oldalán). Például:

```
Console.Write("Alap (Ft): ")
Beolvas = Console.ReadLine()
Alap = CSng(Beolvas)
```

Törtszámok beírásánál a területi beállításoknak megfelelő elválasztójelet (például tizedesvesszőt) használjuk!

A **Szia** program bekéri a felhasználó nevét, majd köszön neki.

Függvények a kifejezésekben

A matematikai függvények elé ki kell írni a *Math.* minősítést[5], például: *Math.Abs*(*x*). Egy kifejezés négyzetgyökét a *Math.Sqrt*(*x*), egészrészét pedig az *Int*(*x*) függvény határozza meg.

Az *Rnd()*, a *Randomize()* és a sztringfüggvények szintaxisa, illetve használata megegyezik a Visual Basic Script azonos függvényeivel.

A **Függvény** program bemutatja a függvények használatát.

Egyszerűsíthetjük a forráskódot, ha a modul legelső sorában (még a *Module* utasítás előtt) elhelyezzük az *Imports System.Math* utasítást. Ennek hatására nem kell kiírnunk a *Math.* minősítést a matematikai függvények alkalmazásánál:

```
Imports System.Math
Module Module1
    Dim Szám As Single
    Sub Main()
        Console.Write("Írj be egy számot: ")
        Szám = CSng(Console.ReadLine())
        Console.WriteLine("A szám abszolút értéke: " & Abs(Szám))
        Console.ReadLine()
    End Sub
End Module
```

Az *Imports* utasítás beírásánál is segítséget kapunk az intelligens súgótól.

[5] A *Math* osztály metódusait hívjuk meg.

A matematikai függvényeket sok más függvénnyel és eljárással együtt a .NET keretrendszer tartalmazza. A több ezer azonosító rendszerezéséhez a keretrendszer elemeit halmazokba, úgynevezett névterekbe csoportosították. (Egy névtéren belül az azonosítóknak egyedinek kell lenniük, de két különböző névtérben szerepelhetnek egyforma elnevezések.) A matematikai függvények például a *Math* névtérben helyezkednek el. Az *Imports* utasítás közli a fordítóprogrammal, hogy az előforduló azonosítókat a *Math* névtérben is keresse. Ezt úgy fejezzük ki, hogy importáltuk a *Math* névteret a programunkba. Importálás nélkül a névtér megjelölését (*Math*) ki kell írunk az azonosító elé: *Math.Abs(x)*.

Feltételes elágazások programozása (116. oldal)

A beolvasás ellenőrzése

A feltételes elágazás (*if … else … end if*) szintaxisa és használata megfelel a Visual Basic Scriptnek. A **Körzet** projekt programja megvizsgálja, hogy a körzetszám egyenlő-e 30-cal.

Feltételek az elágazásnál

A relációk, logikai műveletek, többirányú elágazások szintaxisa és használata megfelel a Visual Basic Scriptnek.

Az ékezetes karaktereket tartalmazó sztringek összehasonlítására Visual Basic Scripthez hasonlóan használhatjuk az *StrComp* függvényt. Egyszerűbb megoldás azonban, ha a forráskód legelejére beírjuk az

```
Option Compare Text
```

utasítást. Ekkor a relációk figyelembe veszik az ékezetes karakterek helyét az ábécében. Ne felejtsük el, hogy ebben az esetben az összehasonlításnál a kisbetűket nem különböztetjük meg a nagybetűktől!

A **Díjak** projekt programja kiszámolja a percdíjakat.

Szövegdobozok, parancsgombok és társaik
A grafikus felület programozása (119. oldal)

A grafikus felület kezelésével kapcsolatban lásd még a *Windows alkalmazás létrehozása* leckét a *Visual Basic 2005 Express Edition – Első lépések* dokumentumban!

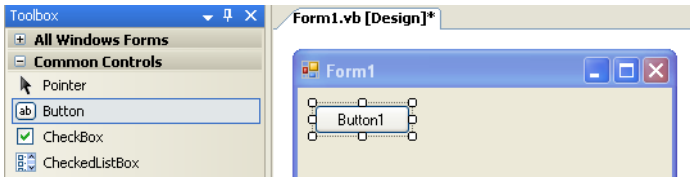
A grafikus felhasználói felület kényelmes lehetőséget biztosít az adatok beolvasásához, a programok futásának vezérléséhez. A szövegdobozok, parancsgombok és más grafikus elemek nagymértékben egyszerűsítik a programok kezelését. Ezeket az elemeket objektumként kezelhetjük.[6]

Objektumok a képernyőn

A képernyőn megjelenő elemek (szövegdobozok, parancsgombok stb.) úgynevezett **objektumok**. Ez azt jelenti, hogy a programokban hivatkozhatunk a tulajdonságaikra, sőt, módosíthatjuk is ezeket a tulajdonságokat. A Visual Basic fejlesztői környezet **vezérőelemeknek** (*controls*) nevezi a képernyőn megjeleníthető objektumokat.

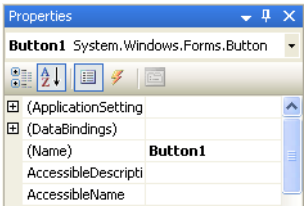
Indítsuk el a Visual Basic fejlesztői környezetét, majd hozzunk létre egy Windows-alkalmazást. A kódszerkesztő ablak helyett most egy úgynevezett tervezőablakot (*Design*) látunk, amelyben megjelenik a készülő program ablaka (*Form1*).[7]

A leggyakoribb vezérlőelemeket a *Common Controls* (általános vezérlők) eszközkészletben (*Toolbox*) találjuk. Helyezzünk el a programablakban egy parancsgombot (*Button*)!



Parancsgomb elhelyezése a készülő programablakban

A hivatkozáshoz azonosítóval kell ellátnunk az objektumokat. Az objektumok azonosítójára ugyanolyan szabályok vonatkoznak, mint a programok változóinak elnevezésére. Az objektum azonosítóját a tulajdonságok (*Properties*) munkalapon adhatjuk meg (*Name* tulajdonság). A tulajdonság neve zárójelben szerepel, hogy a lista elejére kerüljön.[8]



Egy objektum (parancsgomb) név-tulajdonsága

A fejlesztői környezet automatikusan elnevezi a programablakba kerülő objektumokat. Az első parancsgomb például a *Button1* azonosítót kapja. Ezt azonban célszerű az objektum szerepére utaló elnevezésre megváltoztatni. Így könnyebben értelmezhetjük a forráskódot.

Jegyezzük meg, hogy programunk számára maga az ablak is objektum, amely automatikusan a *Form1* nevet kapja a fejlesztői környezettől!

A tulajdonságok (*Properties*) munkalapon az objektumok számos tulajdonságának nevét és értékét megtaláljuk. A munkalap legördülő menüjében választhatjuk ki a módosításra kerülő objek-

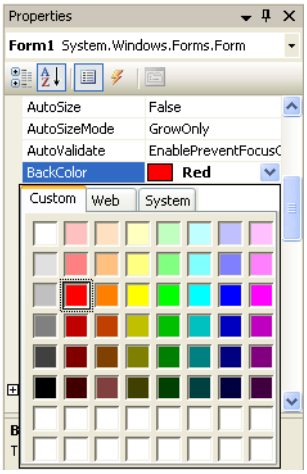
[6] Az objektumok programozása előtt olvassuk el az *Objektumok és események* című bevezetőt!

[7] A Visual Basic terminológiában űrlapnak nevezik a programablakot. Bár az elnevezést nem tartjuk jónak, elterjedtsége miatt mi is alkalmazni fogjuk.

[8] A zárójel ANSI-kódja kisebb, mint a betűk kódja.

tumot. A tulajdonságokat nem csak a program írása közben (tervezésidőben), hanem az utasítások segítségével futás közben (futásidőben) is módosíthatjuk.

Lássunk egy példát a tervezésidejű beállításra! A programablak háttérszínét a *BackColor* tulajdonság szabja meg. Keressük meg a *Form1* objektum tulajdonságai között, majd a legördülő menü segítségével adjunk meg másik színt! Célszerű a *Custom* panelről választani. (A *System* panelen a *Vezérlőpult* alapján beállított megjelenéshez illeszkedő színeket találjuk.)



A háttérszín módosítása

Az objektumok tulajdonságait gyakran **attribútumoknak** nevezzük.

Eseménykezelő eljárás létrehozása

Az objektumelvű programozás egyik következménye, hogy az objektumokkal tevékenységeket végeztethetünk. Az objektummal végrehajtható tevékenységeket az objektum **metódusainak** nevezzük. A metódusok különböző algoritmusokat valósítanak meg, és tetszőleges utasításokat tartalmazhatnak.

Az objektumok a tevékenységeket valamilyen esemény bekövetkezésekor végzik el. Ez az esemény lehet a program indítása vagy például egy egérkattintás a parancsgombra. Az esemény bekövetkezésekor végrehajtandó utasításokat a programon belül külön csoportba, eseménykezelő eljárásba írjuk. Az **eseménykezelő eljárás** utasításait a program az esemény bekövetkezésekor hajtja végre.

Kattintsunk duplán a létrehozott parancsgombra. A fejlesztői környezet az eddig látott tervezőablak (*Design*) mellett megnyitja a már ismert kódszerkesztő ablakot. A kódszerkesztő ablak tartalmazza a készülő Windows-alkalmazás forráskódjának keretét. A forráskód a *Form1* nevű objektumosztályt fogja definiálni. Az osztálydefiníció a *Class osztálynév* és az *End Class* utasítások között helyezkedik el. A *Public* kulcsszó arra utal, hogy osztályunkat más modulok is felhasználhatják.

A kódszerkesztő automatikusan létrehozta a készülő eseménykezelő eljárás keretét. Az eljárás utasításait a *Sub* … *End Sub* utasítások közé kell elhelyezni. A *Sub* kulcsszó a szubrutin (alprogram, eljárás) rövidítése. A *Private* kulcsszó arra utal, hogy eljárásunkat más modulok nem érhetik el.

Az eljárás azonosítója célszerűen az objektum és az esemény megnevezéséből áll, melyeket aláhúzásjel választ el egymástól: *Button1_Click*. Az eljárás paraméterekkel is rendelkezik, melyeket most nem részletezünk.[9] A zárójelben lévő paraméterlista után a *Handles* kulcsszó definiálja, hogy mely objektum mely eseményét fogja kezelni eljárásunk. Mindezekkel egyelőre semmi dolgunk, mert **a fejlesztői környezet automatikusan létrehozza a forráskódnak ezt a részét**.

[9] Lásd az *Első lépések* című dokumentumot!

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, _
                      ByVal e As System.EventArgs) Handles Button1.Click

    End Sub
End Class
```

Az eseménykezelő eljárás kerete

A tulajdonságok futásidejű módosítása

Írjuk meg az eseménykezelő eljárást úgy, hogy ha a felhasználó rákattint a parancsgombra, akkor az ablak háttérszíne váltson kékre!

A forráskódban az objektum egy tulajdonságára az objektum azonosítójával és a tulajdonság megnevezésével hivatkozunk. Az azonosító és a tulajdonságnév közé pontot teszünk. Ezt a formát **minősített hivatkozásnak** nevezzük. Az ablakobjektumra a forráskódban kivételesen nem a nevével, hanem a *Me* kulcsszóval kell hivatkozni!

Írjuk be a *Sub* és *End Sub* utasítások közé a következő sort:

```
Me.BackColor = Color.Blue
```

Futtassuk a programot. Kattintsunk a parancsgombra, és figyeljük meg az eredményt.

A **Kattint** projekt bemutatja a háttérszín módosítását parancsgomb segítségével.

INPUT-objektumok

A grafikus felhasználói felületen általában szövegdobozokkal végezzük az adatok beolvasását, és címkeobjektumok segítségével jelenítjük meg az eredményeket. Ezeknek az objektumoknak a használatát az *Első lépések* című dokumentumban ismertetjük (Windows-alkalmazás létrehozása).

A továbbiakban konzol- és Windows-alkalmazásként is bemutatjuk a példaprogramokat (lásd a *Példa\Konzol*, illetve *Példa\Windows* mappát).

Telefonköltség (122. oldal)

A feltételes ciklusok szintaxisa és használata Visual Basicben pontosan megfelel a Visual Basic Scriptnek. A ciklusok példaprogramjait konzolalkalmazásként elkészítve lásd az **Összeg**, **Hátul**, és **Kilépési** projektben. Az *Összeg* programban használtuk a += értékmódosító utasítást.

Az összegezés algoritmusa

Eseményvezérelt programozás esetén a telefonköltség meghatározásához nincs szükség ciklusra. Helyezzünk el az ablakban egy szövegdobozt, egy parancsgombot, és egy címkeobjektumot. Ha a parancsgombra kattintunk, akkor hozzáadjuk a szövegdobozba írt számot az eddigi összeghez, és a címkeobjektum segítségével megjelenítjük az összeget. Az összeget tároló változó nullázását a deklaráláskor végezzük el.

Az összegezés algoritmusát az **Összeg** projekt mutatja be.

Ciklusok

A ciklusok bemutatásához készítsünk programot, amely meghatározza, hogy 3-nak melyik az a hatványa, amely már nagyobb, mint 1000. Az utasításokat egy *Start* gomb *Click* eseménykezelő eljárásába írjuk bele.

A feladat megoldásához kiindulunk a 3 első hatványából, és addig szorozzuk 3-mal, amíg 1000-nél nagyobb számot nem kapunk. Ha felhasználjuk, hogy az első hatvány biztosan nem nagyobb 1000-nél, akkor alkalmazhatunk hátultesztelő ciklust:

```
Hatványok
    Hatvány = 3
    CIKLUS amíg Hatvány ≤ 1000
        Hatvány = Hatvány · 3
    CIKLUS VÉGE
    Ki: Hatvány
Hatványok vége
```

Megoldás elöltesztelő ciklussal

```
Hatványok
    Hatvány = 3
    CIKLUS
        Hatvány = Hatvány · 3
    AMÍG Hatvány ≤ 1000
    CIKLUS VÉGE
    Ki: Hatvány
Hatványok vége
```

Megoldás hátultesztelő ciklussal

A feltételes ciklusok szintaxisa és használata Visual Basicben pontosan megfelel a Visual Basic Scriptnek. A ciklus lehet elöl- vagy hátultesztelő, mindkét típusnál alkalmazhatunk ismétlési, illetve kilépési feltételt.

A ciklusok különböző típusait a **Hatványok1, Hatványok2, Hatványok3** és **Hatványok4** Windows-alkalmazás mutatja be. A programokban felhasználtuk az értékmódosító utasítást.

További példaként kössük össze az összegezés algoritmusát ciklus szervezésével! 1-től kezdve hány egymást követő természetes számot kell összeadni ahhoz, hogy az összeg elérje a 100-at? A számot és az összeget tartalmazó változót először nullázzuk. A ciklusban a számot eggyel megnöveljük, majd hozzáadjuk az összeghez.

```
Összeadás
    Szám = 0
    Összeg = 0
    CIKLUS AMÍG Összeg < 100
        Szám = Szám + 1
        Összeg = Összeg + Szám
    CIKLUS VÉGE
    Ki: Szám
Összeadás vége
```

Egymást követő természetes számok összege

A megoldást az **Összeadás** Windows-alkalmazás mutatja be.

Összetett adattípusok (125. oldal)

Tömbök a programokban

A tömbök használata Visual Basicben nagyon hasonlít a Visual Basic Scripthez, csak a deklarációban meg kell adni a tömbelemek típusát.[10]

A tömb elemeinek indexelése nullával kezdődik. A deklarációban megadjuk a maximális index értékét és a tömbelemek típusát:

```
Dim tömbnév(index_maximális_értéke) As elemtípus
```

Például: `Dim Névsor(32) As String`

A példában szereplő *Névsor* tömb 33 elemet tartalmaz, melyeket 0-tól 32-ig sorszámozunk. A tankönyv példáiban nem fogjuk használni a 0 indexű tömbelemeket. Ezzel szemléletesebbé tesszük az algoritmusokat. A hétköznapi életben is 1-től kezdjük a sorszámozást.

A tömb elemeire a tömb nevével és az elem zárójelbe tett indexével hivatkozunk: `Névsor(14)`

Értékadás a tömb elemeinek

A deklaráció során a tömb elemeinek értéket is adhatunk:

```
Dim tömbnév() As elemtípus = { elem0, elem1, … }
```

Például:

```
Dim Névsor() As String = { "", "Laci", "Kati", "Orsi" }
```

Figyeljünk a pontos szintaxisra! A tömbelemek megadása esetén nem szerepelhet a deklarációban a maximális index értéke, de a kerek zárójelet ekkor is ki kell írni! A felsorolásban ne feledkezzünk meg a 0 indexű elemről (itt üres sztringként adtuk meg)!

A tömbök használatát a **Beolvas** és **Telefonkönyv** konzolalkalmazás mutatja be.

Eseményvezérelt programozás esetén több érték beolvasásához nincs szükség ciklusra. Helyezzünk el az ablakban egy szövegdobozt, egy parancsgombot és egy címkeobjektumot. Ha a parancsgombra kattintunk, akkor hozzáfűzzük a szövegdobozba írt nevet az eddigi listához, azaz a címkeobjektum *Text* tulajdonságához. Soremelést az előre definiált *vbNewLine* sztringkonstans hozzáfűzésével végezhetünk. Ha betelt a tömb, akkor üzenetet küldünk a felhasználónak.

A felhasználónak az *MsgBox*[11] eljárás segítségével küldhetünk üzenetet. Szintaxisa:

```
MsgBox(üzenet [,parancsgombok [, címsor]])
```

Az *üzenet* paraméter az ablakban megjelenő szöveget adja meg. Ha csak egy OK-gombot szeretnénk megjeleníteni, akkor a *parancsgombok* paraméter helyére írjuk be a *vbOKOnly* előre definiált konstanst (ez el is hagyható). A *címsor* az üzenetablak címének a szövegét határozza meg.

Például:

```
MsgBox("Betelt a tömb!", vbOKOnly, "Hiba")
```

[10] A Visual Basicben a tömbök valójában objektumok. Ezért a *New* záradékkal is létrehozhatjuk a tömböt.

[11] Az *MsgBox* valójában függvény, amely eljárásként is hívható. A függvény visszatérési értéke megadja a felhasználó által lenyomott parancsgomb kódját. Ha eljárásként hívjuk, akkor a visszatérési érték elvész.

Az üzenet megjelenítése

A tömbök használatát a **Beolvas** és a **Telefonkönyv** Windows-alkalmazás mutatja be.

Számlálós ciklusok (128. oldal)

Számlálós ciklusok

A számlálós ciklusok szintaxisa és használata Visual Basicben pontosan megfelel a Visual Basic Scriptnek. A ciklusok példaprogramjait konzolalkalmazásként elkészítve lásd a **Telefonkönyv**, **Négyzetek**, **Páros**, **Vissza, Lottó** és **Megszámlál** projektben.

A ciklusszámláló általában olyan változó, amelyet csak a cikluson belül alkalmazunk, máshol nincs rá szükségünk. Ezért a Visual Basic lehetővé teszi a ciklusszámláló deklarálását a ciklusfejben:

```
For változónév As típus = kezdőérték To végérték [Step lépésköz]
```

Ekkor azonban a cikluson kívül nem használhatjuk a ciklusváltozó szerepét betöltő változót![12]

A ciklusfejben történő változódeklarációt a **Vissza** projekt mutatja be.

Egy tömb elemeinek maximális indexét a *GetUpperBound* függvénnyel határozhatjuk meg.[13] Szintaxisa:

```
tömbnév.GetUpperBound(0)
```

A *GetUpperBound* függvény használatát a **Megszámlál** projektben mutatjuk be. Vegyük észre, hogy sem a tömb deklarációjában, sem a ciklusváltozó végértékénél nem írtuk be konkrétan a maximális index értékét!

A számlálós ciklus alkalmazásai

Mivel a **Telefonkönyv** program Windows-alkalmazásként történő megvalósításánál folyamatosan kiírtuk az ablakba a listát, ezért nincs szükségünk ciklusszervezésre. A ciklusok alkalmazását a **Négyzetek** és a **Páros** projekt mutatja be.

A **Vissza** program elkészítésénél sem volt szükségünk ciklusra. Figyeljük meg a forráskódban, hogyan végezzük el a fordított sorrendben történő kiírást!

Egymásba ágyazott ciklusok

A lottószámok beolvasásánál szintén nincs szükségünk ciklusra. A hibás értékre üzenettel figyelmeztethetjük a felhasználót (például *MsgBox*). Lásd a **Lottó** projektet. Figyeljük meg az egymásba ágyazott elágazásokat!

Az egymásba ágyazott ciklusok bemutatásához készítsünk programot, amely kiírja a 7 első 50 többszörösét. Egy sorba kerüljön 5 szám, így 10 sort kell megjelenítenünk. A többeseket ciklusok segítségével rendezzük sorokba és oszlopokba.

[12] A ciklusra nézve lokális változó (blokk- azaz itt ciklusszintű hatókör).
[13] A függvény a tömb objektum egy metódusa. Paramétere eggyel kisebb, mint a tömb azon dimenziójának a sorszáma, melynek maximális indexére rákérdezünk (egydimenziós tömböknél tehát 0).

```
Eljárás Hetes
    Többes = 7
    CIKLUS Sor = 1-től 10-ig
        CIKLUS Oszlop = 1-től 5-ig
            Ki: Többes
            Többes = Többes + 7
        CIKLUS VÉGE
        Ki: soremelés
    CIKLUS VÉGE
Eljárás vége
```

A 7 többszörösei

A 7 többszöröseit kiíró programot a **Hetes** Windows-alkalmazás mutatja be. A számokat egy soron belül két-két szóközzel választottuk el egymástól.

Formátumkódok használata

Ha futtatjuk a *Hetes* programot, akkor láthatjuk, hogy nem alakultak ki áttekinthető oszlopok. A formázott megjelenítéshez használjuk a *String.Format* függvényt![14]

A *String.Format* függvény a megadott értékeket formázott karaktersorozattá alakítja. Szintaxisa:

`String.Format(szöveg, érték0, érték1, …)`

ahol a *szöveg* az egyes értékek formátumát tartalmazó sztringkifejezés.
A formátum alakja: {*index* [, *hossz*][:*formátumkód*]}, melynek részei:
index: a kiírásra kerülő érték sorszáma a paraméterlistában (0-val kezdődik a sorszámozás),
hossz: az adott érték számára fenntartott karakterek száma (jobbra zárva az értéket),
formátumkód: a kiírásra kerülő érték formátuma.
A formátumkódok közül itt csak az F-et említjük meg, amelyet a számok fixpontos megjelenítésére használunk. Az F után kell írni a megjelenítendő tizedesjegyek számát, például F0 vagy F3.
Példa a *String.Format* alkalmazására:

```
String.Format("A {0, 5:F1} négyzetgyöke: {1, 10:F4}.", 20, Math.Sqrt(20))
```

A függvény a következő karaktersorozatot alakítja ki:

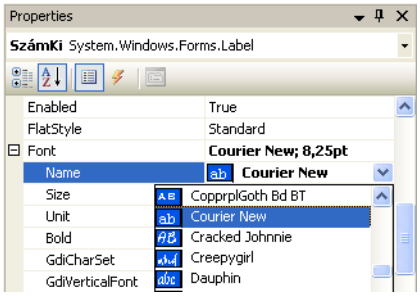
```
A  20,0 négyzetgyöke:     4,4721.
```

A paraméterlista 0. értékét (azaz a 20-at) 5 karakteren jobbra zárva, 1 tizedessel írja ki. A paraméterlista 1. értékét (azaz a 20 négyzetgyökét) 10 karakteren jobbra zárva, 4 tizedessel jeleníti meg.
A *Format* függvény ennél jóval változatosabb formátumok kialakítását teszi lehetővé. Itt azonban nem tárgyaljuk teljes részletességgel.

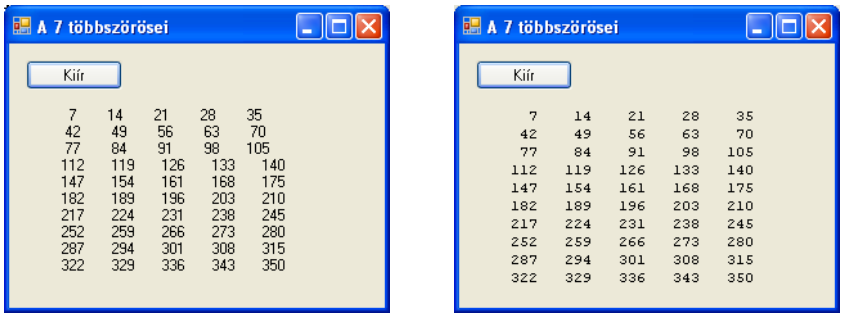
A *String.Format* függvény alkalmazásakor még mindig nem alakulnak ki áttekinthető oszlopok, mert a Visual Basic alapértelmezett betűtípusa a *Microsoft Sans Serif*, amely változó szélességű, úgynevezett proporcionális betűkből áll (az *i* betű szélessége például kisebb, mint az *m* betűé). A tulajdonságok (*Porperties*) munkaablakban váltsuk át a megjelenítést végző címkeobjektum betűtípusát (*Font*/*Name*) például *Courier New*-ra. Így már valóban egymás alá kerülnek az azonos helyiértékek a megjelenítésnél.

A formátumkód használatát a **HetesFormáz** Windows-alkalmazás mutatja be.

[14] A *String* objektumosztály *Format* osztálymetódusa.



A Courier New betűtípus kijelölése a tulajdonságok ablakban



A táblázat megjelenítése MS Sans Serif és Courier New betűtípussal

A megszámlálás algoritmusa

A megszámlálás algoritmusát a **Megszámlál** Windows-alkalmazás mutatja be.

Keresés a tömbben (130. oldal)

A lineáris keresés algoritmusát a **Keres**, a bináris keresését a **Bináris** projekt mutatja be.

Lineáris keresés

Az *Array.IndexOf* függvény[15] egydimenziós tömb esetén elvégzi a keresést. Szintaxisa:

`Array.IndexOf(`*`tömbnév, keresett_érték`*`)`

Az `Array.IndexOf(Név, "Pisti")` függvényhívás például megkeresi a *Név* nevű tömbben a *Pisti* előfordulását.

A függvény visszatérési értéke a keresett érték indexe. Ha a keresett érték nem szerepel a tömbben, akkor a visszatérési érték –1.

Az *Array.IndexOf* függvény alkalmazását az **IndexKeres** projekt mutatja be. A függvény használata esetén nem kell foglalkoznunk a keresési algoritmus megírásával!

[15] Az *Array* objektumosztály *IndexOf* osztálymetódusa. A függvénynek megadhatjuk a tömbelemek összehasonlításának algoritmusát. Lásd a Visual Basic dokumentációjában, illetve a Súgóban.

Keresés rendezett tömbben

Az *Array.BinarySearch* függvény[16] egydimenziós, **rendezett tömb** esetén elvégzi a bináris keresést. Szintaxisa:

```
Array.BinarySearch(tömbnév, keresett_érték)
```

Az `Array.BinarySearch(Név, "Pisti")` függvényhívás például megkeresi a *Név* nevű tömbben a *Pisti* előfordulását.

A függvény visszatérési értéke a keresett érték indexe. Ha a keresett érték nem szerepel a tömbben, akkor a visszatérési érték negatív.[17]

Az *Array.BinarySearch* függvény alkalmazását a **BinárisKeres** projekt mutatja be.

A telefonkönyv rendezése (132. oldal)

A tankönyv példáit a **Bővít**, **Névsor** és **Rendez** projekt mutatja be.

A parancsgombok működésének vezérlése

A Windows-alkalmazásokban célszerű letiltani a beolvasást vezérlő parancsgomb működését, ha elértük a tömb végét. Ezzel megakadályozzuk az indexhatár túllépését.

Egy parancsgomb működését az objektum *Enabled* tulajdonságának *False*-ra állításával tilthatjuk le:

```
Beolvas.Enabled = False
```

Ha a tulajdonság értékét *True*-ra állítjuk, akkor ismét engedélyezzük a parancsgomb kiválasztását.

A lecke példáit bemutató Windows-alkalmazásokban felhasználtuk a parancsgomb működésének letiltását.

Rendezés az Array.Sort eljárással

A Visual Basic *Array.Sort* eljárása[18] elvégzi egy egydimenziós tömb elemeinek rendezését.[19] Szintaxisa:

```
Array.Sort(tömbnév [, kezdőindex, elemszám])
```

A *kezdőindex* és *elemszám* paraméterek használata esetén a tömb megadott része kerül rendezésre. Ha a teljes tömböt rendezzük, akkor figyeljünk az esetlegesen fel nem használt 0 indexű elemre!

Az *Array.Sort* eljárás használatát az **ArraySort** projekt mutatja be.

---

[16] Az *Array* objektumosztály *BinarySearch* osztálymetódusa. A függvénynek megadhatjuk a tömbelemek összehasonlításának algoritmusát. Lásd a Visual Basic dokumentációjában, illetve a Súgóban.

[17] A negatív visszatérési érték bináris komplemense az első olyan index értékét adja meg, amelyhez nagyobb tömbelem tartozik, mint a keresett érték. Ha a bináris komplemens nagyobb, mint a tömb legnagyobb indexe, akkor a tömb nem tartalmaz a keresettnél nagyobb értéket. Az index bináris komplemensét például az `index Xor -1` kifejezéssel határozhatjuk meg.

[18] Az *Array* objektumosztály *Sort* osztálymetódusa. Az eljárásnak megadhatjuk a tömbelemek összehasonlításának algoritmusát. Lásd a Visual Basic dokumentációjában, illetve a Súgóban.

[19] A rendezéshez a quicksort (gyorsrendezés) algoritmust használja.