SZÉCHENYI ISTVÁN EGYETEM

EXCEL PROGRAMOZÁS

Oktatási segédlet

Pusztai Pál

2014

Tartalomjegyzék

1. BEVEZETÉS

Jelen dokumentum elsősorban az Excel programozás iránt (kedvtelésből vagy egyéb okból) érdeklődő „kezdőknek" készült, akik most teszik meg első lépéseiket egy olyan úton, amelynek során, ill. végén megvalósul az ember és számítógép kommunikációjának legmélyebb, legnehezebben elsajátítható, de egyben legintellektuálisabb szintje, a számítógép programozása.

Ha egy feladat megoldására számítógépes programot készítünk, akkor azt általában nem úgy tesszük, hogy elkezdjük begépelni a program utasításait. Minél nagyobb, összetettebb a megoldandó feladat, annál inkább szükséges a következő megoldási lépések végrehajtása:

1. A feladat megfogalmazása, pontosítás, általánosítás.
2. Matematikai (vagy egyéb) modell kiválasztása, megadása (ha szükséges, illetve lehetséges).
3. Az adatszerkezet definiálása, az input-output specifikálása.
4. A megoldást megvalósító algoritmus megtervezése, elkészítése.
5. Programírás, kódolás (az adatszerkezet és az algoritmus alapján).
6. Tesztelés, hibakeresés.
7. Dokumentálás (felhasználóknak, fejlesztőknek).

Természetesen az adott feladat (vagy munka) jellegéből adódóan bizonyos lépések el is maradhatnak (pl. 2., 7.), illetve javítás, módosítás esetén szükség lehet egy korábbi szintre való visszalépésre is. A program tényleges megírása (5.) csak a már elkészített megoldó algoritmus (4.) ismeretében lehetséges, ami viszont nem készülhet el anélkül, hogy tisztáznánk, milyen bemenő és milyen eredmény adataink lesznek (3.).

Az Excel programozási segédletünket a programíráshoz (5.) szükséges (rövidített) nyelvi ismertetővel kezdjük (lásd 2. fejezet), amit a programok kipróbálásához (6.) szükséges ismeretek követnek (lásd 3. fejezet). A harmadik részben (lásd 4. fejezet) az Excel objektumairól, azok programból történő használatáról lesz szó. Mivel a segédlet bevezető jellegű, ezért a feldolgozott témakörnek csak egy kisebb (de reményeink szerint a lényeges dolgokat tartalmazó) szeletét mutatjuk be.

A szakirodalom szerencsére bőségesen ellát minket programozásról szóló szakkönyvekkel. Az Excel programozás után érdeklődő olvasóknak az [1], a Visual Basic nyelvvel kapcsolatosan a [2], az algoritmusok tervezését illetően kezdőszinten a [3], haladóbb szinten a [4] szakirodalmat ajánljuk.

2. A VISUAL BASIC PROGRAMOZÁSI NYELV

A BASIC (Beginner's All-purpose Symbolic Instruction Code) programnyelvet oktatási célokra hozták létre 1964-ben (Dartmouth College USA, Kemény János és Thomas Kurtz). Az általános célú felhasználhatóság és a könnyű tanulhatóság elsődleges szempont volt, ez a mozaikszó szavaiból is kiderül.

A BASIC programozási nyelv magas szintű, az emberi gondolkodáshoz, jelölésrendszerhez közel álló programozási nyelv. A BASIC nyelven megírt programokat (az ún. forrásprogramokat) a számítógép nem tudja egyből végrehajtani. Ezek végrehajtásához fordítás (compile) vagy értelmezés (interpretation) szükséges. A fordítóprogramok a teljes forrásprogramot lefordítják a számítógép által már végrehajtható utasításokká, míg az értelmezők utasításonként értelmezik és hajtják végre a forrásprogramot.

A nyelvnek többféle változata létezett, kezdve az iskola-számítógépek beépített BASIC értelmezőjétől, a 80-as években elterjedt személyi számítógépek Qbasic-jén keresztül, az 1991-ben megjelent Microsoft Visual Basic (röviden VB) nyelvig. A Visual Basic for Applications (röviden VBA) az MS Office szoftverek makrónyelve, a Visual Basic Script a Windows operációs rendszer scriptnyelve, a 2002-ben megjelent Visual Basic .NET pedig a .NET keretrendszer programozási nyelve.

A Visual Basic nyelv után ejtsünk pár szót a nyelvet használó szoftverfejlesztő rendszerekről is.

Az MS Visual Studio egy általános célú szoftverfejlesztő keretrendszer, amelyben többféle (pl. Visual Basic, C#) programnyelven is fejleszthetünk.

Az MS Visual Basic for Applications (pl. az általunk használt Excel VBA) főbb tulajdonságai:

- Csak Visual Basic nyelven programozhatunk, (a Visual Studio-hoz képest) korlátozott fejlesztési eszköztár mellett.
- Csak az adott szoftverrel (pl. Excel) együtt használható.
- Önállóan futtatható (*.exe) fájlok nem készíthetők.
- Az adott szoftverhez igazodó, „készen kapott" objektumrendszert tartalmaz.

Az egyes utasításoknál megadjuk az utasítások (esetleg egyszerűsített) szintaktikáját. Az egyszerűsítéssel a lényeges dolgok kiemelése a célunk, az utasítások teljes szintaktikája a súgóban megtalálható.

Megjegyzés

- A dokumentumban az MS Excel 2010 VBA segítségével szemléltetünk, a leírások is ehhez igazodnak, de a 2003-es Excel VBA környezete ugyanúgy alkalmas a tanulásra, a feladatok megoldására.
- Az utasítások szintaktikájában a szögletes zárójelben lévő részek elhagyhatók, a kapcsos zárójelek között, függőleges vonallal elválasztva választási lehetőségek felsorolása található, a három pont a tetszőleges számú ismétlés jelölésére szolgál.

2.1. Egyszerű adattípusok

Minden programozási nyelv (így a VB is) meghatározza az adatok azon körét, amelyet kezelni tud. Azt, hogy milyen fajta adatokat használhatunk, ezekkel milyen műveleteket végezhetünk, ezek hogyan tárolódnak, az *adattípusok* definiálják. Attól függően, hogy az adattípus egy vagy több logikailag összetartozó adat használatát engedi meg, megkülönböztetünk *egyszerű* és *összetett* adattípusokat. Az egyszerű adattípusokról ebben a fejezetben, az összetett adattípusokról a 2.5. fejezetben lesz szó.

2.1.1. Egész adattípusok

Az egész számok használatát többféle egész típus biztosítja, amelyek az adatok tárolására felhasznált memóriaterület méretében, az előjel kezelésében, így az egyes típusokhoz tartozó egész számok tartományában különböznek.

Adattípus	Tárolási méret	Tartomány
`Byte`	1 bájt	0 .. 255
`Integer`	2 bájt	−32768 .. 32767
`Long`	4 bájt	−2147483648 .. 2147483647

2.1. táblázat. Az egész adattípusok

Míg a `Byte` típus 1 bájtja csupán 2^8=256 db, addig a `Long` típus 4 bájtja már 2^{32} (kb. 4 milliárd) különböző érték (egész szám) tárolását biztosítja.

Az egész típusú adatokkal a matematikában szokásos műveletek végezhetők el, amelyet a 2.2. és 2.3. táblázatok szemléltetnek. Az aritmetikai műveletek táblázatbeli sorrendje a műveletek erősorrendjét (prioritás, precedencia) tükrözi, ahol legelöl a legerősebb (legmagasabb prioritású) hatványozás található.

A hasonlítások között nincs erősorrendbeli különbség (azonos prioritásúak), de prioritásuk gyengébb, mint az aritmetikai műveleteké. A hasonlítási műveletek nemcsak egész számokra, de más adatokra (pl. valós számok, sztringek, stb.) is értelmezettek, eredményük logikai típusú. A logikai adattípusról a 2.1.3. fejezetben, míg a kifejezések kiértékelésének szabályairól a 2.2.2. fejezetben lesz szó.

Aritmetikai műveletek
Hatványozás (^)
Negáció (−)
Szorzás (*), osztás (/)
Egész osztás hányadosa (\)
Egész osztás maradéka (`Mod`)
Összeadás (+), kivonás (−)

2.2. táblázat. Az egész adattípusok aritmetikai műveletei

Hasonlítások
Egyenlő (=)
Nem egyenlő (<>)
Kisebb (<)
Nagyobb (>)
Kisebb vagy egyenlő (<=)
Nagyobb vagy egyenlő (>=)

2.3. táblázat. A hasonlítási műveletek

Pl. −3^2 → −9 3/2 → 1.5 5\3 → 1 5 `Mod` 3 = 2 → `True`

Megjegyzés

- A kisebb vagy egyenlő (<=), illetve a nagyobb vagy egyenlő (>=) műveletek (ahogyan azt a nevük is sugallja) csak akkor igazak, ha a kisebb (<) vagy egyenlő (=), illetve a nagyobb (>) vagy egyenlő (=) műveletek legalább egyike igaz.
- A műveleteket operátoroknak, a műveletekben résztvevő adatokat operandusoknak, a hasonlításokat pedig relációs műveleteknek (vagy egyszerűen csak relációknak) is nevezik.

2.1.2. Valós adattípusok

A valós számok esetén kétféle típust használhatunk, amelyek jellemzőit a 2.4. táblázat szemlélteti. Noha a használható számok nagyságrendje kb. 10^{38}, a tárolásra használt 4, illetve 8 bájt csak kb. 7-8, illetve 15-16 értékes (decimális) számjegyet biztosít (a többi számjegy a kerekítés miatt 0 lesz).

Adattípus	Tárolási méret	Értékes jegyek
`Single`	4 bájt	7-8
`Double`	8 bájt	15-16

2.4. táblázat. A valós adattípusok

A valós adattípusok műveletei az egész osztás (\, `Mod`) műveletek kivételével megegyeznek az egész adattípusok műveleteivel.

Pl. 4^−0.5 → 0.5

2.1.3. Logikai adattípus

A logikai (`Boolean`) adattípusban kétféle érték létezik, az igaz (`True`) és a hamis (`False`). A három legfontosabb logikai művelet a *tagadás* (`Not`), az *és* (`And`), és a *vagy* (`Or`), amelyek igazságtáblázatát a 2.5. táblázat szemlélteti. A logikai értékekre értelmezettek a hasonlítás műveletek (lásd 2.3. táblázat) is.

A	B	`Not` A	A `And` B	A `Or` B
`True`	`True`	`False`	`True`	`True`
`True`	`False`	`False`	`False`	`True`
`False`	`True`	`True`	`False`	`True`
`False`	`False`	`True`	`False`	`False`

2.5. táblázat. Logikai műveletek

A *tagadás* egyoperandusú művelet az ellenkezőjére változtatja a logikai értéket (igazból hamis lesz és fordítva), az *és* kétoperandusú művelettel összekapcsolt logikai kifejezés csak akkor lesz igaz, ha mindkét operandus igaz, míg a *vagy* kétoperandusú művelettel összekapcsolt logikai kifejezés csak akkor lesz hamis, ha mindkét operandus hamis.

Megjegyzés

- Logikai művelet még a *kizáró vagy* (`Xor`), az *ekvivalencia* (`Eqv`), és az *implikáció* (`Imp`) is.
- A logikai értékek között is értelmezett a sorrendiség (bár ezt ritkán használjuk), nevezetesen a `False` érték megelőzi a `True` értéket, azaz `False` < `True` → `True`, így a logikai értékekre definiált az összes hasonlítási művelet.

2.1.4. Szöveges adattípus

A szöveges (`String`) adattípus szöveges adatok (karaktersorozatok) használatát biztosítja. A `String` kulcsszó változó hosszú sztringeket deklarál, amelyek a maximális adathosszig (2^31 darab karakter) tetszőleges hosszúak lehetnek. A sztringeket macskakörmök közé kell tenni. Az üres sztringnek ("") nincs egyetlen karaktere sem.

A sztringekre az *összefűzés* művelet (más néven *konkatenáció*) (+, &), és a hasonlítások (lásd 2.2. táblázat) értelmezettek. A + művelet csak szövegeket fűz össze, az & számokat is képes összefűzni, amelyeket a művelet elvégzése előtt sztringgé alakít (konvertál).

Pl. "alma" + "fa" → "almafa" 3 & 3 * 5 → "315"

A hasonlítási műveletek kiértékelése a sztringek karakterei alapján történik. Két sztring egyenlő (=), ha egyforma hosszúak és karaktereik rendre megegyeznek, egyébként nem egyenlők (<>). A kisebb (<), illetve nagyobb (>) hasonlítási műveleteknél a sztringek első különböző karakterpárja határozza meg az eredményt (lásd megjegyzés). Ha nincs ilyen karakter (az egyik sztring kezdőszelete a másiknak), akkor a rövidebb sztring lesz a kisebb.

Pl. "Alma" < "alma" → `True`
"Kovács" < "Kovácsné" → `True`
"Kovacs" < "Kovács" → `True`

Megjegyzés

- A sztringek összehasonlítására az `Option Compare` (modulszintű) utasítás is hatással van.
  - Az `Option Compare Binary` (alapértelmezett) esetben a karakterek (Windows kódlapbeli) kódjai alapján történik a hasonlítás (`A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø`).
  - Az `Option Compare Text` utasítás olyan összehasonlítást eredményez, amely nem különbözteti meg a kis- és nagybetűket (case insensitive) (`(A=a) < (À=à) < (B=b) < (E=e) < (Ê=ê) < (Z=z) < (Ø=ø)`).
- Lehetőség van fix hosszú sztringek használatára is. Ekkor a `String` kulcsszó után (egy csillag karakterrel elválasztva) megadandó a sztringek hossza is (pl. `String * 30`). Egy ilyen típusú változóban tárolt sztring hossza fixen a típusban rögzített hossz lesz (a példában 30), a hosszabb sztringek jobbról csonkulnak, a rövidebbek pedig szóközökkel egészülnek ki. (A változókról a 2.2.1. fejezetben lesz szó, a fix hosszú sztringek használatáról egy példát a 2.5.2. és 2.5.3. fejezetekben láthatunk.) Bizonyos esetekben (pl. véletlenelérésű adatfájloknál) csak ez a sztringtípus használható. A maximális adathossz 2^16 darab karakter.
- A sztringek minta alapján történő hasonlítására a `Like` művelet használható.

2.1.5. Egyéb adattípusok

A dátum/idő (`Date`) adattípus segítségével dátum és/vagy időadatokat tudunk kezelni. Egy (8 bájton tárolt) valós szám egész része a dátum, tört része pedig az időpont megadására használatos. Az időkezelés megegyezik az Excel időkezelésével, a dátumkezelés kicsit eltérő. Az Excel csak pozitív számokat tud dátumként értelmezni, a VB negatív számokat is kezel, amellyel a dátum 100.01.01-től 9999.12.31-ig terjedő érték lehet.

Pl. Excel: 1.25 ~ 1900.01.01 6:00:00, VB: 1.25 ~ 1899.12.31 6:00:00, −1.5 ~ 1899.12.29 12:00:00

A `Currency` adattípus a pénzügyi számításokhoz ajánlott. Egy ilyen típusú adat fixen négy tizedes jegyet tartalmaz, mert az adat tízezerszerese tárolódik egész számként (8 bájton, ami 19-20 értékes decimális jegy pontosságot jelent).

A `Variant` adattípust akkor használjuk, ha egy adatnak nem tudjuk előre a típusát, vagy egy változóban különböző típusú adatokat (pl. szám, szöveg) szeretnénk tárolni. A változókról a 2.2.1. fejezetben lesz szó. A kényelmes használat ára a nagyobb memóriaterület (szám esetén 16 bájt, szöveg esetén 22 + a szöveg karaktereinek számával megegyező bájt).

Megjegyzés: A `Variant` típusú változók speciális értékeket (`Empty`, `Error`, `Nothing`, `Null`) is tartalmazhatnak.

A VBA előszeretettel használja a felsorolt (`Enum`) típust akkor, amikor az adatoknak csak néhány lehetséges értéke van. Az `Enum` típusú adatok mindegyikéhez egy azonosító és egy egész szám rendelhető. A forrásprogramokban ugyan mindkettő használható, de célszerű az azonosítók használata, így a forráskód kifejezőbb, érthetőbb lesz.

Szintaktika:

```
[Private|Public] Enum name
membername[=constantexpression]
membername[=constantexpression]
...
End Enum
```

`Private`	A típus csak abban a modulban hivatkozható, amelyikben deklaráltuk.
`Public`	A típus minden modulból hivatkozható (ez az alapértelmezés).
name	A típus azonosítója.
membername	A típushoz tartozó elem azonosítója.
constantexpression	A típushoz tartozó elem értéke.

Az egyes elemek számértékeit megadó kifejezések (*constantexpression*) konstansokból álló, egész értékű kifejezések lehetnek. Ezek a kifejezések elhagyhatók, ekkor az első elem esetén 0, a többinél az előző értéknél eggyel nagyobb érték definiálódik.

Pl.

```
Enum SecurityLevel
  IllegalEntry = -1
  SecurityLevel1 = 0
  SecurityLevel2 = 1
End Enum
```

Megjegyzés

- Az `Enum` és `End Enum` utasítások közötti rész (példában szereplő) beljebb tagolása csak az áttekinthetőséget segíti.
- Az `Enum` utasítás modulszintű utasítás, azaz a modulok elején helyezendő el. A modulok felépítéséről a 3.2.5. fejezetben lesz szó.
- Az egyes utasítások szintaktikáját a VBA fejlesztőkörnyezet súgója alapján adjuk meg.

2.2. Adatok kezelése

Ahhoz, hogy adatainkat a számítógép kezelni tudja, tárolnia is kell. A tárolás miként, konkrét megvalósítása egyrészt az adatok típusától, másrészt az alkalmazott fejlesztőkörnyezettől, és az operációs rendszertől is függ.

Adatainkat alapvetően a számítógép memóriájában tároljuk, de szükség esetén külső adathordozón (adatfájlokban) is eltárolhatjuk. A fájlokban történő adattárolásra akkor van szükség, ha adatainkat két programfutás között is meg szeretnénk őrizni. A memóriában történő adattárolásról ebben a fejezetben, az Excel-fájlokban (munkafüzetekben) történő adattárolásról a 4.1. fejezeteben lesz szó.

2.2.1. Változó

Változón olyan azonosítóval ellátott memóriaterületet értünk, ahol a változó típusának megfelelő értéket (pl. adatot, részeredményt) tárolhatunk. Egy változóban tárolt érték a program végrehajtása során megváltozhat – innen ered az elnevezése –, ilyenkor a változóba kerülő új érték felülírja a régit.

A változók használatát általában megelőzi azok deklarálása, amikor is megadjuk a változó típusát. A legtöbb programozási nyelvben (pl. C, Pascal) kötelező a változók deklarálása, de a Visual Basic megengedi a változók deklaráció nélküli használatát is. A deklarálatlan változók típusa `Variant` lesz.

Mindazonáltal a deklarálatlan változók (a „kényelmes” használat mellett) lehetséges hibaforrások is egyben (pl. egy változó azonosítójának elgépeléséből adódó hiba csak futásidőben derül ki), ezért a VB külön utasítást biztosít arra, hogy a változókat deklarálni kelljen. Az `Option Explicit` (modulszintű) utasítás

kikényszeríti a változók deklarálását azáltal, hogy szintaktikai (formai) hibát kapunk egy nem deklarált változó használatakor (lásd 3.2.6. fejezet).

A változók deklarálásának (egyszerűsített) szintaktikája:

```
Dim varname [As type] [,...]
```

varname	A változó (betűvel kezdődő) azonosítója (neve).
type	A változó típusa (ha hiányzik, a változó `Variant` típusú lesz).

Pl.

```
Dim i As Integer          'Egy Integer típusú változó deklarálása
Dim v                     'Egy Variant típusú változó
Dim a,b As Single         'Egy Variant és egy Single típusú változó
```

Megjegyzés

- A példában szereplő sorok végén magyarázó megjegyzések találhatók, amelyeket a Visual Basic Editor (lásd 3.2.4. fejezet) alapértelmezetten zöld színnel emel ki.
- Egy változó azonosítójának (mint minden más programbeli azonosítónak) be kell tartaniuk a VB névmegadási szabályait (naming rules). Ezek többek között előírják, hogy betűvel kell kezdődnie, nem haladhatja meg a 255 karaktert, nem tartalmazhat speciális karaktereket (pl. ., !, @, &, $, #), stb. Célszerű olyan beszédes (azaz a változó szerepére, a benne tárolt adatra/adatokra utaló), rövid, alfanumerikus karaktersorozatot használni, amelyeknek nincs más jelentésük a VB-ben.
- Az azonosítókban magyar ékezetes betű és az alulvonás karakter (_) is használható (pl. Év_Hó_Nap), de a kis és nagybetűk között nincs különbség. Egy változó azonosítója a deklaráláskor megadott formában jelenik meg hivatkozáskor (pl. ha `i`-t deklaráltunk, akkor az `I`-vel való hivatkozás `i`-re cserélődik a kódszerkesztő ablakban).
- Noha a deklarált változóknak a VB ad kezdőértéket (a numerikus változók 0, a logikai változók `False`, míg a `String` típusú változók üres sztring kezdőértéket kapnak), lehetőleg csak olyan változók értékeit használjuk fel, amelyeknek korábban már definiáltuk az értékét!

Típusdeklarációs karakterek

A VB nyelv (hasonlóan, mint a korábbi BASIC nyelvek) megengedi azt, hogy a nem deklarált változók típusát típusdeklarációs karakterrel jelezzük. Ezek használatának jelentősége csökkent, hiszen a VB-ben kikényszeríthetjük a változók deklarálását, amikor is explicit módon meg kell adnunk a változó típusát.

A használható karakterek és a hozzájuk tartozó adattípusok:

% `Integer`, ! `Single`, # `Double`, $ `String`, @ `Currency`

Megjegyzés

- Az első értékadásnál megadott típusdeklarációs karakter később el is hagyható.
- A változók típusa a Deftype utasításokkal (pl. `DefInt`, `DefStr`, …) is szabályozható.

Pl.

```
i% = 2.8                  'Az i változóba 3 kerül (kerekítés)
c@ = 123456789012#        'A c változóba egy Double konstanst teszünk
st$ = 2                   'Az st változóba "2" kerül
st = st + st              'Az st változóba "22" kerül
```

Megjegyzés: A fenti példák az értékadó utasítást használják, amiről részletesen a 2.2.4. fejezetben lesz szó.

2.2.2. Kifejezés

Kifejezésen olyan számítási műveletsort értünk, amellyel megmondjuk, hogy milyen adatokkal, milyen műveleteket, milyen sorrendben kívánunk elvégezni. A kifejezés kiértékelésekor egy új érték − a kifejezés értéke − keletkezik.

A kifejezésben szerepelhetnek:

- Konstansok, változók, függvényhívások
- Műveletek
- Zárójelek

Pl. `(-b+Sqr(b*b-4*a*c))/(2*a)`

A példában a `4` és `2` konstansok, az `a`, `b`, `c` változók, az `Sqr` a négyzetgyök függvény.

A műveletek prioritása csökkenő erősorrendben:

- Aritmetikai
  - Hatványozás (^)
  - Negáció (−)
  - Szorzás, osztás (*, /)
  - Egész osztás hányadosa, maradéka (\, `Mod`)
  - Összeadás, kivonás (+, −)
- Szöveg összefűzés (&, +)
- Hasonlítások (=, <>, <, >, <=, >=, `Like`, `Is`)
- Logikai (`Not`, `And`, `Or`, `Xor`, `Eqv`, `Imp`)

Az egyes hasonlítások azonos prioritásúak, az egész osztás műveleteit és a logikai műveleteket csökkenő prioritás szerinti sorrendben adtuk meg.

A kifejezések kiértékelésének szabályai:

- A zárójelbe tett kifejezések és függvényhívások operandus szintre emelkednek.
- A magasabb prioritású műveletek végrehajtása megelőzi az alacsonyabb prioritású műveletek végrehajtását.
- Az azonos prioritású műveleteknél a balról-jobbra szabály érvényes, ami azt jelenti, hogy ezen műveletek végrehajtása balról jobbra haladva történik.

Megjegyzés

- Ha szükséges, akkor a típuskonverziók automatikusan végrehajtódnak.
- Az `Is` művelettel objektumhivatkozások egyezése vizsgálható.

Pl.

```
3 * 4 <= 12 And "a" & 12 = "a12" → True
3.8 \ 2 * 3 → 0
(3.8 \ 2) * 3 → 6
```

Az első példában a szorzás, majd az összefűzés (a `12` szöveggé konvertálásával), utána a hasonlítások, végül az `And` művelet hajtódik végre, amivel `True` értéket kapunk. A második és harmadik példában szereplő `3.8` valós szám egy egész osztásban szerepel, ezért értéke egészre konvertálódik (`4`-re kerekítődik). A második példában először a szorzás (eredménye 6), majd az egész osztás (`4\6`) hajtódik végre, így az eredmény `0` lesz. A harmadik példa zárójelezése megváltoztatja a prioritásból adódó sorrendet, így először az egész osztás (`4\2`) hajtódik végre, majd a szorzás, így az eredmény `6` lesz.

2.2.3. Függvények

A programozási nyelvek beépített függvényekkel segítik a számolást, adatfeldolgozást. Ezeket csak használnunk kell, azaz a megfelelő paraméterekkel meg kell hívnunk őket. Az alábbiakban (csoportokba foglalva) felsorolunk néhány gyakran használatos függvényt (de hangsúlyozzuk, hogy ez a felsorolás korántsem teljes, a VB-ben sokkal több függvényt használhatunk).

Matematikai függvények

`Abs(X)`	`X` abszolút értéke.
`Exp(X)`	Az exponenciális függvény (e^x) értéke az `X` helyen.
`Log(X)`	A természetes alapú logaritmus függvény értéke az `X` helyen.
`Sin(X)`	`X` szinusza (`X` radiánban adott).
`Cos(X)`	`X` koszinusza (`X` radiánban adott).
`Sqr(X)`	`X` négyzetgyöke.
`Int(X)`	A legnagyobb egész szám, amely még nem nagyobb, mint `X`.
`Rnd()`	Egy véletlen szám a [0, 1) intervallumból.

Pl.

```
Int(3.8) → 3          Int(-3.8) → -4
```

Egy véletlen egész szám az `[a,b]` intervallumból: `Int((b-a+1)*Rnd)+a`

Megjegyzés: A paraméterek nélküli függvényeknél az üres zárójelpár elhagyható (mint a példában az `Rnd` esetén).

Konverziós függvények

`Asc(X)`	Az `X` karakter ASCII kódja.
`Chr(X)`	Az `X` ASCII kódú karakter.
`Str(X)`	Az `X` numerikus adat szövegként.
`Val(X)`	Az `X` számot tartalmazó szöveg numerikus értéke.

Pl.

```
Asc("A")→65    Chr(65)→ "A"    Str(2.3)→" 2.3"    Val("2.3")→ 2.3
```

Adott típusú értékké konvertáló függvények

`CStr(X)`	`X` értékét `String` értékké.
`CInt(X)`	`X` értékét `Integer` értékké.
`CSng(X)`	`X` értékét `Single` értékké.
`CDate(X)`	Az `X` érvényes dátumkifejezést `Date` értékké.

Pl.

```
CStr(2.6)→ "2,6"      CInt(2.6) → 3      CSng("2,6") → 2.6
```

Megjegyzés

- Ezekből a konvertáló függvényekből csak néhányat ragadtunk ki, de minden egyszerű adattípushoz létezik ilyen függvény.
- A kékkel kiemelt függvénynevek (csakúgy, mint a többi, kék színnel kiemelt szó) kulcsszavak, nem használhatók másra (pl. egy változó azonosítójának).
- Az átalakítandó `X` érték tetszőleges típusú érték lehet (pl. szám, szöveg, logikai érték).
- A `CStr` függvény az operációs rendszerbeli tizedesjelet használja (a példa azt az esetet szemlélteti, amikor a beállított tizedesjel a vessző).

- Ha számot tartalmazó szöveges adatot konvertálunk numerikus adattá, akkor tizedesjelként a vessző és az operációs rendszerben beállított tizedesjel használható. Ha pl. vessző az operációs rendszerben beállított tizedesjel, akkor a pont használata esetén típuskeveredési hibát kapunk (pl. `CInt("2.6")`).
- Ha az `X` érték nem konvertálható az eredménytípus értékkészletébe, akkor hibát kapunk (pl. `CByte(-1)`).

Szövegkezelő függvények

`Len(X)`	Az `X` sztring hossza (karaktereinek száma).
`Left(X,Y)`	Az `X` sztring elejéről `Y` darab karakter.
`Right(X,Y)`	Az `X` sztring végéről `Y` darab karakter.
`Mid(X,Y[,Z])`	Az `X` sztring `Y`-adik karakterétől `Z` darab karakter.
`Trim(X)`	Az `X` sztring vezető és záró szóközeinek levágása.
`InStr([X,]Y,Z)`	A `Z` sztring megkeresése az `Y` sztringben (az `X`-edik karaktertől kezdődően).

Pl.

```
Len("Alma")→ 4          Left("Alma",2)→ "Al"          Right("Alma",2)→ "ma"
Mid("Alma",3,1)→ "m"    Mid("Alma",3)→ "ma"           Trim(" a b ")→ "a b"
InStr("alma","a")→ 1    InStr(2,"alma","a")→ 4        InStr("alma","A")→ 0
```

Megjegyzés

- Vezető szóközök levágása: `LTrim`, záró szóközök levágása: `RTrim`.
- Ha a `Mid` függvény `Z` paraméterét nem adjuk meg, vagy megadjuk, de nincs annyi darab karakter az `Y`-adik karaktertől kezdődően, akkor az eredmény az `X` sztring utolsó karakteréig tart.
- Az `InStr` függvény `X` paramétere elhagyható, ekkor a keresés az `Y` sztring első karakterétől indul.
- Létezik `Mid` utasítás is, amellyel egy sztring karakterei más karakterekre cserélhetők.

Dátum- és időkezelő függvények

`Date`	Az aktuális (operációs rendszerbeli) dátum.
`Time`	Az aktuális (operációs rendszerbeli) idő.

Megjegyzés: A dátumok hasonlóan kezelhetők, mint az Excel-ben (pl. a `Date-1` kifejezés a tegnapi dátumot adja).

Adatformázó függvény

`Format(X[,Y])`	Az `X` kifejezés sztringgé alakítása az `Y` formázó sztring alapján.

Pl.

```
Format(100/3,"0.00") → "33,33"
Format(CDate("89.12.05"), "Long Date") → "1989. december 5."
```

2.2.4. Az értékadó utasítás

Az értékadó utasítással egy változónak adhatunk értéket. Az utasítás szintaktikája:

```
[Let] varname = expression
```

varname	A változó azonosítója.
expression	A tárolandó értéket meghatározó kifejezés.

Az utasítás végrehajtásakor először kiértékelődik az értékadás jobb oldalán álló kifejezés (*expression*), azaz kiszámítódik a kifejezés értéke, majd ha a kapott érték tárolható az adott változóban (*varname*), akkor megtörténik a tárolás, különben futási (run-time) hiba lép fel.

Megjegyzés

- Az utasítás kulcsszava (`Let`) elhagyható, ezért többnyire el is hagyjuk.
- A számolás közben, illetve a tárolás előtt a VB implicit (általunk nem definiált) típuskonverziókat hajthat végre. A megfelelő konverziós függvények használatával azonban explicit (általunk megadott) módon szabályozhatjuk a szükséges típuskonverziókat, így elkerülhetjük az implicit típuskonverziókat és az esetlegesen ebből eredő programhibákat.

Pl.

```
Dim i As Byte           'Az i változó deklarálása
i = 255                 'A legnagyobb Byte érték
i = i + 1               'Túlcsordulási hiba (Overflow)!
v = "szöveg"            'Variant típusú változóba szöveget
v = 3.14                'Variant típusú változóba számot
i = "a"                 'Típuskeveredési hiba (Type mismatch)
i = "2" + "2"           'Az i változóba 22 kerül
i = "2" + "2" * "2"     'Az i változóba 6 kerül
```

A példában egy `Byte` típusúra deklarált `i` és egy deklarálatlan (ezért `Variant` típusú) `v` változónak adunk különböző értékeket.

Az első értékadás a legnagyobb `Byte` típusú adatot teszi az `i` változóba. Ezzel nincs semmi baj, de ekkor a második értékadás túlcsordulási (overflow) futási hibát eredményez, hiszen a kifejezés értéke 256 lesz, ami már nem tárolható az `i` változóban. Ha az `i` változó aktuális értéke (tartalma) nem 255, akkor az utasítás rendben végrehajtódik, azaz az `i` változóban eggyel nagyobb szám lesz, mint az értékadás előtt.

A harmadik és negyedik értékadás azt szemlélteti, hogy egy `Variant` típusú változóba különböző típusú adatokat is tehetünk.

Az ötödik értékadás egy típuskeveredési (type mismatch) futási hibát ad, hiszen egy (számmá nem alakítható) szöveget szeretnénk tárolni egy numerikus változóban.

Az utolsó két értékadás az automatikus típuskonverziót szemlélteti. Az egyik értékadás kifejezésének eredménye 22, hiszen két db sztringet fűzünk össze, de az `i` változóba ennek a sztringnek az egész számmá alakított értéke kerül. A másik értékadásban a szorzás művelet hajtódik végre először (a nagyobb prioritás miatt), ezért a szorzás operandusai számmá konvertálódnak, majd összeszorzódnak (4). Mivel a + művelet egyik operandusa numerikus, ezért ez a művelet az összeadás művelet lesz (és nem a sztringek összefűzése), ami szintén numerikussá alakítja a bal oldali operandusát (`"2"`). Eredményül tehát a 6 érték számolódik ki, és tárolódik az `i` változóban.

2.2.5. Adatok bekérése

Egy változónak nemcsak az előbb ismertetett értékadó utasítással adhatunk értéket. Lehetőség van a változók értékeinek a program futása (végrehajtása) során történő megadására is. Például egy másodfokú egyenletet megoldó program esetén célszerű az egyenletet meghatározó együtthatókat bekérni, ahelyett, hogy azok konkrét értékeit a programba beírnánk, hiszen a bekérő utasítás használatával a program módosítása nélkül tudunk különböző másodfokú egyenleteket megoldani. Elegendő csak újra futtatnunk a megoldó programot és más bemenő adatokat adni.

Az alábbi VB függvény egy párbeszédablak segítségével egy szöveges adat bekérését, megadását biztosítja. Az utasítás (egyszerűsített) szintaktikája:

```
InputBox(prompt[,title][,default])
```

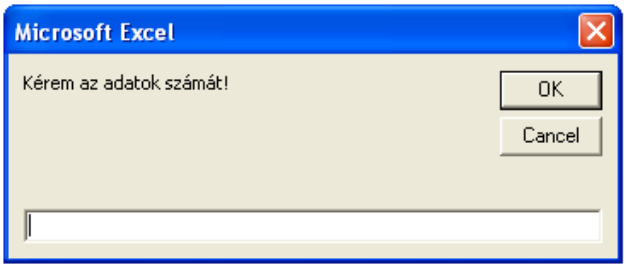
prompt	Tájékoztató üzenet.
title	Az ablak fejlécében megjelenő szöveg.
default	Az adat alapértelmezett értéke.

Pl.

```
Dim n As Integer
n = InputBox("Kérem az adatok számát!")
```

A példa egy egész szám bekérését szemlélteti, ahol csak a kötelező (*prompt*) paramétert adtuk meg. A bekért szöveges adatot egy numerikus változóba (n) tesszük (implicit típuskonverzióval), ezért rossz adat (pl. nem számmá alakítható szöveg) megadása esetén hibaüzenetet kapunk.

A függvény végrehajtásakor egy párbeszédablak jelenik meg (lásd 2.1. ábra). Az adatot a beviteli mezőben kell megadni. A függvény eredménye az OK gomb megnyomása (vagy az Enter billentyű leütése) esetén a megadott szöveg lesz, egyébként pedig (Cancel gomb, Esc billentyű, ablak bezárás) az üres sztring.



2.1. ábra. A példában szereplő InputBox függvény párbeszédablaka

Megjegyzés

- A függvénynek egyéb opcionális paraméterei is vannak.
- Valós számok bekérése esetén ügyeljünk arra, hogy az implicit típuskonverzió tizedesjelként csak a vesszőt és az operációs rendszerben beállított tizedesjelet fogadja el. Ha pl. vessző az operációs rendszerben beállított tizedesjel, akkor a pont használata esetén típuskeveredési hibát kapunk.

2.2.6. Adatok kiírása

Többnyire még a legegyszerűbb programoknak is vannak bemenő (input) és eredmény (output) adatai. A bemenő adatok egyik megadási lehetőségét az előzőekben ismertettük, az eredmények megjelenítési lehetőségeiről most lesz szó.

Az alábbi VB függvény egy párbeszédablak segítségével egy szöveges adatot jelenít meg. Az utasítás (egyszerűsített) szintaktikája:

```
MsgBox(prompt[,buttons][,title])
```

prompt	A kiírandó adat.
buttons	Az ablak nyomógombjait definiáló érték.
title	Az ablak fejlécében megjelenő szöveg.

Pl.

```
MsgBox("2*3=" & 2*3)
MsgBox "2*3=" & 2*3
MsgBox 3^2,,"Három négyzete"
```

Az első két esetben a kiírandó adatot két adat (egy szöveg és egy szám) összefűzésével állítottuk elő, a harmadik esetben nem adtuk meg a *buttons* paramétert. A megfelelő párbeszédablakok a 2.2. ábrán láthatók (az első két MsgBox hívás ugyanazt eredményezi).

2.2. ábra. A példában szereplő MsgBox függvények párbeszédablakai

Megjegyzés

- A kiírandó adat tetszőleges típusú kifejezéssel megadható, ekkor a kifejezés értéke (implicit típuskonverzióval) sztringgé konvertálódik. Valós számok tizedesjele az operációs rendszerben beállított tizedesjel lesz.
- Ha a *buttons* paramétert elhagyjuk, akkor csak az OK gomb jelenik meg. A paraméterrel nemcsak a megjelenő nyomógombok, de a párbeszédablak egyéb tulajdonságai is definiálhatók.
- A függvénynek egyéb opcionális paraméterei is vannak.
- Noha a MsgBox függvény, a függvény visszatérési értékét csak akkor használjuk, ha be kell azonosítani, hogy melyik gombbal zárják be az ablakot.
- A példában szereplő MsgBox függvényt eljárásként hívtuk meg. A szubrutinokról és azok hívási szabályairól (pl. zárójel kirakás/elhagyás) a 2.4.1. fejezetben lesz szó.

Ha sok adatot kell megjelenítenünk, akkor az egyenkénti megjelenítés esetén minden adat után be kell zárnunk a megjelenő párbeszédablakot. Alkalmazva az előző példában szereplő gondolatot, célszerű az összes megjelenítendő adatot egy sztringgé összefűzni, így elegendő egyetlen MsgBox hívás az adatok megjelenítésére. Ennél a megoldásnál az adatokat a `Chr(13)` (carriage return), illetve `Chr(10)` (linefeed) karakterek segítségével többsoros szöveggé is alakíthatjuk. Mivel a MsgBox által kiírt szöveg max. 1024 db karakterből állhat, ezért nagy mennyiségű adat kiírására inkább az alábiakban ismertetésre kerülő megoldást használjuk.

Az adatmegjelenítés történhet a `Debug` objektum `Print` metódusának segítségével is, amely a Visual Basic Editor (lásd 3.2. fejezet) Immediate ablakába (lásd 3.2.1. fejezet) ír ki. Az objektumokról a 2.6. fejezetben, az Excel objektumairól a 4.1. fejezetben lesz szó.

Szintaktika:
```
Debug.Print [outputlist]
```

outputlist	A kiírandó kifejezést vagy kifejezéseket megadó lista.

A lista egy elemének (kifejezésének) megadási szintaktikája:
```
[{Spc(n)|Tab(n)}] expression charpos
```

`Spc(`*n*`)`	*n* db szóköz kiírása (opcionális).
`Tab(`*n*`)`	A kiírás az *n*–edik oszlopban kezdődjön (opcionális).
expression	A kiírandó kifejezés (opcionális).
charpos	A következő kiírandó adat kezdőpozíciójának megadása (opcionális).

Pl.
```
Debug.Print ("2*3=" & 2*3)
Debug.Print "2*3="; 2*3; Tab(10); "3*4=”; 3*4
```

Megjegyzés

- A metódushívásokra a szubrutinok hívási szintaktikája érvényes (lásd 2.4.1. fejezet).
- Ha a *charpos* pontosvessző, akkor az aktuális sorban folytatódik a kiírás, egyébként meg a következő sor elején, így egy `Debug.Print` hívás utolsó adata után kirakott vagy elhagyott pontosvesszővel szabályozható, hogy a következő `Debug.Print` hol kezdje a kiírást.
- Ha a kiírás már meghaladta a `Tab(`*n*`)` által megadott oszlopot, akkor a következő sor *n*-edik oszlopában folytatódik a kiírás.
- A kiírandó adatok vesszővel is elválaszthatók.
- A `Print` metódus az operációs rendszerbeli tizedesjelet használja.
- Az Immediate ablak (lásd 3.2.1. fejezet) tartalma szerkeszthető. Szerkesztéskor ügyeljünk arra, hogy a kiírás majd attól a helytől kezdődik el, ahol a kurzor állt, így könnyen összekeveredhetnek az egyes futások által kiírt adatok. Célszerű tehát az ablakban lévő szöveg végére állni (pl. Ctrl+End), vagy törölni az ablak teljes tartalmát (pl. Ctrl+A, Del), mielőtt elindítunk egy programot, amely az Immediate ablakba ír.
- Az ismertetett utasításokon kívül más eszközök is rendelkezésünkre állnak arra, hogy a programunk adatokat kapjon, illetve adatokat jelenítsen meg. A 3.2.7. fejezetben vizuális vezérlők segítségével, míg a 4.1.3. fejezetben Excel munkafüzet segítségével valósul meg a felhasználói input/output.
- A VB általános fájlkezeléséről nem lesz szó, de azt megemlítjük, hogy a fájlkezelő utasítások segítségével adatainkat fájlba menthetjük, illetve onnan visszatölthetjük, így két különböző futás között is megőrizhetjük adatainkat.

2.3. Vezérlőszerkezetek

Az eddigiekben szó volt a változók deklarálásáról, az értékadó utasításról, az adatbekérő és adatkiíró lehetőségekről, így ezekből már összeállíthatunk egy olyan programot, amely bizonyos bemenő adatokból valamilyen eredményadatokat számol, és ezeket meg is jeleníti. A megfelelő utasításokat ilyenkor egyszerűen sorban egymás után (szekvenciálisan) megadjuk, illetve végrehajtatjuk.

De mi van akkor, ha egy utasítást csak egy adott feltétel teljesülése esetén kell végrehajtani, vagy ha egy utasítást többször is végre szeretnénk hajtani? Szükségünk van olyan eszközökre, amelyekkel az egyes tevékenységek végrehajtási sorrendje előírható. Ezeket az eszközöket *vezérlőszerkezeteknek* nevezzük.

A strukturált programozás három vezérlőszerkezet használatát engedi meg, ezek a *szekvencia*, a *szelekció* és az *iteráció*. Bizonyítottan minden algoritmus megadható ezekkel a vezérlőszerkezetekkel, ezért a feladatok megoldásánál mi is csak ezeket fogjuk használni.

Szekvencia:	Az utasítások egymás után, a megadásuk sorrendjében hajtódnak végre.
Szelekció:	A végrehajtandó utasítás(ok) feltétel(ek)től függően kerül(nek) kiválasztásra.
Iteráció:	Egy vagy több utasítás ismételt végrehajtása.

Megjegyzés

- A strukturáltság és a forrásprogramok ehhez igazodó tagolása áttekinthetőbb, ezáltal könnyebben karbantartható programokat eredményez.
- A VB megenged nem strukturált programokat eredményező, speciális vezérlésátadó utasításokat (pl. `GoTo`, `Exit`) is, de ezekkel nem foglalkozunk.

2.3.1. Szekvencia

A szekvencia vezérlőszerkezetben a program utasításai (algoritmus esetén ez egyes lépések, tevékenységek) egymást követően, az utasítások megadási sorrendjében hajtódnak végre. A VB-ben az egyes utasításokat általában külön sorokba írjuk, de egy sorba több utasítást is írhatunk, ekkor az utasítások közé kettőspontot kell tenni.

Pl.

```
'Utasítások megadása egy sorban (végrehajtás: balról jobbra haladva)
Dim i As Integer: i = 3 * 5: MsgBox (i)

'Utasítások megadása egymást követő sorokban (végrehajtás: fentről lefelé)
Dim i As Integer
i = 3 * 5
MsgBox (i)
```

Megjegyzés: A kétféle megadás tetszés szerint, vegyesen is alkalmazható.

2.3.2. Szelekció

A szelekció vezérlőszerkezet segítségével a végrehajtandó utasítások között szelektálhatunk, azaz megadhatjuk, hogy mikor melyik utasítás, illetve utasítások hajtódjanak végre. Az alábbiakban azt a szelekciós utasítást ismertetjük, amelyben a végrehajtandó utasítások (*statements*) logikai kifejezésekkel (*condition*) választhatók ki.

Az `If` utasítás szintaktikája:

```
If condition Then [statements] [Else elsestatements]
```

vagy

```
If condition Then
  [statements]
[ElseIf condition-n Then
  [elseifstatements]]
...
[Else
  [elsestatements]]
End If
```

Az első esetben az egész utasítás egy sorba kerül (ekkor nem kell `End If` az utasítás végére), míg a második esetben több sorba (ahol az egyes ágakban lévő utasítások beljebb tagolása nem kötelező, de javítja az áttekinthetőséget).

Végrehajtás:

- Az első teljesülő feltételhez tartozó utasítások kerülnek végrehajtásra.
- Ha egyik feltétel sem teljesül és az `Else` ág definiált, akkor ezek az utasítások (*elsestatements*) kerülnek végrehajtásra.

Pl.

```
If d = 0 Then
  MsgBox ("Nulla")
ElseIf d > 0 Then
  MsgBox ("Pozitív")
Else
  MsgBox ("Negatív")
End If
```

Megjegyzés

- Az egyes ágakban újabb `If` utasítások is lehetnek, azaz az `If` utasítások egymásba ágyazhatók.
- A VB másik szelekciós utasítása a `Select Case` utasítás, amely egy (általában nem logikai, hanem pl. egész, szöveg) kifejezés értéke alapján választja szét az egyes ágakat. Az utasítást nem részletezzük, de megjegyezzük, hogy minden `Select Case` utasítás felírható `If` utasítás segítségével is.

2.3.3. Iteráció

Az iteráció vezérlőszerkezet segítségével egy vagy több utasítás ismételt végrehajtása valósítható meg. Azokat az utasításokat, amelyeket az iteráció (vagy más néven ciklus) ismételten végrehajt, ciklusmagnak nevezzük. Három alapvető ciklusfajta létezik: az *elöltesztelő,* a *hátultesztelő* és a *növekményes* ciklus.

Az elöltesztelő ciklus a ciklust vezérlő feltételt a ciklusmag végrehajtása előtt vizsgálja, a hátultesztelő ciklus pedig a ciklusmag végrehajtása után. A növekményes ciklus egy speciális elöltesztelő ciklus, amely a ciklusmagot egy változó adott értékei mellett hajtja végre.

Mivel a ciklusszervezés a programozás egyik alapvető eleme, ezért a magas szintű programozási nyelvek (így a VB is) mind a három ciklusfajtát támogatják. Ez lehetővé teszi, hogy az adott feladathoz legjobban illeszkedő ciklusfajtát válasszuk, amivel a megoldásunk (programunk) érthetőbb és áttekinthetőbb lesz, de megjegyezzük, hogy az elöltesztelő ciklussal a másik két ciklusfajta is megvalósítható.

A VB ciklusszervező utasításait ismertetjük a továbbiakban.

A `While` ciklus szintaktikája:

```
While condition
  [statements]
Wend
```

Végrehajtás:

- Amíg a ciklust vezérlő feltétel (*condition*) igaz, addig végrehajtódnak a ciklusmag utasításai (*statements*).
- A feltételt mindig a ciklusmag végrehajtása előtt vizsgálja (elöltesztelő ciklus), ezért lehet, hogy a ciklusmag egyszer sem kerül végrehajtásra (ekkor *üres ciklusról* beszélünk).

Megjegyzés: A ciklusmag utasításainak hatással kell lennie a feltétel igazságértékére (ez a `Do` ciklusra is érvényes), különben a ciklus üres ciklus, vagy végtelen ciklus (ekkor a Ctrl+Break vagy az Esc billentyűkkel leállítható a programfutás).

Pl.

```
i = 1
While i <= 3
  MsgBox i: i = i + 1
Wend
```

A `For` ciklus szintaktikája:

```
For counter=start To end [Step step]
  [statements]
Next [counter]
```

Végrehajtás:

- A ciklusmag (*statements*) végrehajtási feltétele:
  - *counter<=end*, ha *step*>=0
  - *counter>=end*, ha *step*<0
- A feltételt mindig a ciklusmag végrehajtása előtt vizsgálja (elöltesztelő ciklus), ezért lehet, hogy a ciklusmag egyszer sem kerül végrehajtásra.
- A ciklusváltozó (*counter*) a ciklusmag minden egyes végrehajtásakor a *step* értékével megváltozik. A *step* megadása nem kötelező, ha elhagyjuk, akkor az alapértelmezett értéke 1.

Pl.

```
For i = 1 To 5 Step 2
  MsgBox i
Next
```

Megjegyzés: A VB-ben létezik egy olyan speciális ciklusutasítás is (`For Each`), amelyben a ciklusváltozó egy gyűjtemény objektumain „lépdel végig" (lásd 4.2.4. fejezet).

A `Do` ciklus szintaktikája:

```
Do [{While|Until} condition]
  [statements]
Loop
```

vagy

```
Do
  [statements]
Loop [{While|Until} condition]
```

Végrehajtás:

- A ciklust vezérlő feltétel (*condition*) vizsgálata az első esetben a ciklusmag (*statements*) végrehajtása előtt (elöltesztelő ciklus), a második esetben pedig utána (hátultesztelő ciklus) történik meg.
- `While` kulcsszó esetén, ha a feltétel igaz, akkor a ciklusmag (ismételt) végrehajtása következik, egyébként a ciklusból való kilépés.
- `Until` kulcsszó esetén, ha a feltétel igaz, akkor a ciklusból való kilépés történik, egyébként a ciklusmag (ismételt) végrehajtása.

A `Do` ciklusutasítás tehát egy olyan általános ciklusszervező utasítás, amellyel mind az elöltesztelő, mind a hátultesztelő ciklusfajta megvalósítható, ráadásul úgy, hogy a feltétellel megadhatjuk az ismétlés, illetve a kilépés feltételét is.

Megjegyzés

- A ciklusmag utasításainak beljebb tagolása nem kötelező, de az áttekinthetőség érdekében célszerű.
- Az `If` utasításhoz hasonlóan a ciklusok is egymásba ágyazhatók.
- Vegyük észre a `For` ciklus *step*=0 esetét, és a `Do` ciklust vezérlő feltétel elhagyhatóságát! Ekkor a ciklusokból való kilépés az `Exit For`, `Exit Do` utasítások segítségével történhet, amelyeket a strukturáltság megőrzése érdekében nem említettünk (de a VB megengedi a használatukat).

2.4. Szubrutinok

Szubrutinon egy jól meghatározott, önálló tevékenységsort megvalósító, formailag is elkülönülő programrészt értünk. A szubrutinok tényleges végrehajtásához általában egy másik, a szubrutint aktivizáló, azt *meghívó* programrész is szükséges. Egy szubrutin többször is meghívható, így annak szolgáltatásai a program megfelelő helyein egy-egy hívással igénybe vehetők.

A szubrutin a programtervezés, illetve programozás alapszintű, de egyben alapvető fontosságú eszköze. Használatukkal érvényesíthető az *egységbezárási* elv, miszerint egy adott feladat megoldásához szükséges adatoknak és a „rajtuk dolgozó" algoritmusnak egy olyan egysége valósítható meg, amely csak a szükséges mértékben kommunikál a „külvilággal", a szubrutint hívó programrészekkel, egyébként zárt, „dolgait" saját maga „intézi", azokba „nem enged bepillantást".

Megkülönböztetjük a szubrutin *deklarálását,* ahol definiáljuk a szubrutin által végrehajtandó tevékenységeket, a külvilággal való kapcsolattartás módját és szabályait, valamint a szubrutin *hívását,* amikor felhasználjuk a szubrutin szolgáltatásait, vagyis definiálva a kapcsolattartás eszközeit, kiváltjuk a tevékenységsor egy végrehajtását.

A kapcsolattartás legfőbb, de nem kizárólagos (hiszen pl. „mindenhonnan látható" (public) változókon keresztül is történhet a kommunikáció) eszközei a *paraméterek*.

A helyes programozási stílus a *teljes paraméterezésre* való törekvés. Ez azt jelenti, hogy egy szubrutin a működéséhez szükséges adatokat a paraméterein keresztül kapja, és az eredményeket (a függvényérték kivételével) ezeken keresztül adja vissza. Ezzel ugyanis elkerülhető egy esetleges módosítás olyan „mellékhatása", amely a program egy „távoli részének" végrehajtásakor, előre nem látott, nem tervezett változást, s ezzel többnyire nehezen felderíthető hibát okoz.

Megjegyzés

- A VBA-ban megírt összes forrásprogram modulokba, azon belül szubrutinokba szervezett. Egy VBA projekt felépítéséről a 3.2.3. fejezetben lesz szó.
- Az objektumorientált programozás objektumtípusai (osztályai) az adatoknak és a rajtuk dolgozó algoritmusoknak egy még általánosabb egységét valósítják meg azáltal, hogy az adatokhoz nemcsak egy, de tetszőleges számú algoritmus rendelhető. Az algoritmusokat megvalósító szubrutinok (más terminológiával metódusok, tagfüggvények) egyrészt „látják" az objektum adatait, másrészt ők definiálják, írják le, modellezik az objektum viselkedését. Osztályokat nem fogunk definiálni, de az Excel objektumait (lásd 4.1. fejezet) használni fogjuk.

2.4.1. Deklaráció és hívás

Kétféle szubrutint különböztetünk meg, az *eljárást* és a *függvényt*. Az eljárást általában akkor használjuk, ha valamilyen tevékenységet kell végrehajtanunk, a függvényt pedig akkor, ha valamilyen eredményértéket kell kiszámolnunk.

Mivel mind az eljárás, mind a függvény képes adatokat kapni és eredményadatokat visszaadni, így egy adott szubrutin eljárás, illetve függvény alakban is programozható. Az, hogy egy szubrutint eljárás vagy függvény formájában írunk meg, nem elvi kérdés, inkább programozói stílus kérdése. A két forma egymással ekvivalens módon mindig helyettesíthető. Az eljárások és függvények deklarálása és hívása, ha kicsit is, de eltérő.

Eljárás deklarálásának (egyszerűsített) szintaktikája:

```
[Private|Public] Sub name[(arglist)]
[statements]
End Sub
```

Függvény deklarálásának (egyszerűsített) szintaktikája:

```
[Private|Public] Function name[(arglist)] [As type]
[statements]
[name=expression]
End Function
```

`Private`	A szubrutin csak abból a modulból hívható, amelyikben deklaráltuk.
`Public`	A szubrutin minden modulból meghívható (ez az alapértelmezés).
name	A szubrutin azonosítója.
arglist	A szubrutin formális paraméterei (opcionális).
type	A függvény eredményértékének a típusa.

Vegyük észre, hogy a függvény utasításai között szerepel egy olyan értékadó utasítás is, amivel a függvény nevének (*name*) mint változónak adunk értéket. Ez az utasítás szolgál a függvény eredményének a beállítására. Több ilyen utasítás is elhelyezhető a függvény utasításai között, de célszerű egyet elhelyezni, azt is a függvény végén, azaz az `End Function` utasítás előtt.

A formális paraméterek megadásának szintaktikája:
`[Optional][ByVal|ByRef][ParamArray]`*varname*`[()][As` *type*`][=`*defaultvalue*`]`

`Optional`	A paraméter elhagyható.
`ByVal`	A paraméter átadása érték szerinti.
`ByRef`	A paraméter átadása cím szerinti (ez az alapértelmezés).
`ParamArray`	Tetszőleges számú `Variant` típusú paraméter átadásához.
varname	A paraméter azonosítója.
type	A paraméter típusa.
defaultvalue	Az opcionális paraméter alapértelmezett értéke.

A paraméterek neve előtt álló kulcsszavakkal a paraméterek elhagyhatósága, valamint a paraméterek átadási módja definiálható. Az érték szerinti (`ByVal`) paramétereken keresztül a szubrutin csak kaphat adatokat, míg a cím szerinti átadású (`ByRef`) paramétereken keresztül kaphat is, és vissza is adhat eredményeket. Ezért a `ByVal` paramétereket a szubrutin bemenő (input) adatainak megadásához, a `ByRef` paramétereket pedig a szubrutin bemenő és/vagy kimenő (output) adatainak megadásához használhatjuk.

Megjegyzés:

- Ha egy szubrutinnak több paramétere van, akkor a paramétereket vesszővel kell elválasztani.
- Egy `Optional` paramétert csak `Optional` paraméterek követhetnek, azaz egy elhagyható paramétert nem követhet kötelezően megadandó paraméter.
- A `ParamArray` paraméter csak a paraméterlista utolsó eleme lehet. Ilyen paraméter esetén nem használhatunk `Optional` paramétereket.
- Ha egy paraméternek nem adjuk meg a típusát, akkor a paraméter `Variant` típusú lesz.
- A paraméter neve (*varname*) mögötti üres zárójelpár azt jelzi, hogy az átadott paraméter egy tömb (lásd 2.5.1. fejezet). A tömböket csak cím szerinti paraméterátadással lehet átadni.

Eljárás hívása
`[Call]` *name*`[`*argumentlist*`]`

name	A meghívott eljárás azonosítója.
argumentlist	Az aktuális paraméterek.

Az eljárás hívásának kulcsszava (`Call`) elhagyható. Megadásakor az argumentumlistát zárójelbe kell tenni, ha elhagyjuk, akkor a zárójelet is el kell hagyni.

Függvény hívása

A függvények kifejezések részeként hívhatók (hasonlóan, mint a VB függvényei), azaz a függvényhívás bárhol állhat, ahol egy, a függvény eredményének megfelelő érték állhat (pl. egy értékadó utasítás jobb oldalán). Az aktuális paraméterek zárójelei nem hagyhatók el.

Egy szubrutin hívásakor megtörténik az aktuális és formális paraméterek megfeleltetése (a paraméterek átadása), majd a végrehajtás a hívott szubrutin utasításaival folytatódik. A hívott szubrutin befejeztével a végrehajtás a hívást követő utasítással folytatódik.

Hívás megnevezett paraméterekkel

Ez a fajta szubrutinhívás kényelmes hívást biztosít olyan szubrutinok esetén, amelyeknek sok opcionális paramétere van. Elegendő csak azokat a paramétereket és aktuális értékeiket felsorolni, amelyeket át akarunk adni, a többinek meghagyjuk az alapértelmezett értékét.

Pl.

```
ActiveWorkbook.SaveAs FileName:="Mentes.xls"
```

A példában egy objektum egy metódusát hívtuk meg, ami az aktív munkafüzetet menti el a megadott névvel. Az objektumokról és a metódusokról a 2.6., illetve 4.1. fejezetekben lesz szó.

Megjegyzés

- Általában a programnyelvekben kötött a szubrutinok deklarálásának és hívásának sorrendje, nevezetesen a deklarálásnak meg kell előznie (forráskód szinten) a hívást, de a VB megengedi, hogy egy szubrutint a hívó szubrutin után deklaráljunk.
- Egy függvény eljárásként is meghívható (pl. ha a függvény eredményére nincs szükségünk). Ekkor az eljárás hívási szabályai érvényesek. A példáinkban szereplő MsgBox függvényt is eljárásként hívtuk meg a `Call` szó elhagyásával. Az egyetlen paramétert néhol zárójelbe raktuk, de ekkor a zárójelek nem a MsgBox függvény zárójelei, hanem a paraméter egy felesleges zárójelezése.
- Ha egy szubrutint eljárásként hívunk meg egyetlen paraméterrel és elhagyjuk a `Call` szót, akkor a paraméter esetleges (felesleges) zárójelezésével a paraméter átadása érték szerinti lesz.
- A metódushívásokra a szubrutinok hívási szabályai érvényesek.
- A megnevezett paraméterek tetszőleges sorrendben megadhatók. A paraméterek neve és értéke közé a legyen egyenlő (`:=`) jelet, míg a paraméterek közé vesszőt kell tenni.

2.4.2. Mintafeladat

A szubrutinok deklarálását és hívását az alábbi feladat segítségével szemléltetjük.

Feladat: Fordítsunk meg egy adott sztringet!
Pl.”Réti pipitér” → ”rétipip itéR”

Az alábbiakban mind függvénnyel, mind eljárással megoldjuk a feladatot, és mindkét esetben egy hívó, tesztelő szubrutint is készítünk. Erre azért van szükség, mert a paraméterekkel rendelkező szubrutinok nem futtathatók közvetlenül (lásd 3.2.6. fejezet), márpedig a megoldó függvénynek, illetve eljárásnak lesz paramétere. A paramétert a megfordítandó sztring átadására, az eljárással történő megoldás esetén még az eredmény visszaadására is használjuk.

Az első megoldásban (`Fordit` függvény) a kezdetben üres eredménysztringhez (`er`) jobbról hozzáfűzzük a sztring karaktereit fordított sorrendben, azaz először a legutolsót, majd az utolsó előttit, legvégül az elsőt. Amikor az eredménysztring előállt, akkor a megfelelő értékadással (amikor is a függvény neve értéket kap) gondoskodunk az eredmény beállításáról.

A második megoldásban (`MegFordit` eljárás) a kezdetben üres eredménysztringhez (`er`) balról hozzáfűzzük a sztring karaktereit eredeti sorrendben, aminek eredményeként a sztring első karaktere kerül

az eredménysztring legvégére, a második karaktere lesz az utolsó előtti, végül az utolsó karaktere kerül az eredménysztring legelejére.
Természetesen a megoldások lényegét adó ciklusok lehetnének egyformák is, de a kétfajta megközelítéssel azt szerettük volna érzékeltetni, hogy még egy ilyen egyszerű feladatnak is létezhet többféle megoldása.

Megjegyzés: Olyan megoldás is elképzelhető, amelyben a sztring megfelelő karakterpárjait cserélgetjük fel. Az első karaktert az utolsóval kell kicserélni, a másodikat az utolsó előttivel, és így tovább.

```
'Sztring megfordítása függvénnyel
Function Fordit(st As String) As String
Dim i As Integer
Dim er As String
er = ""
For i = Len(st) To 1 Step -1
  er = er + Mid(st, i, 1)
Next
Fordit = er
End Function

'A Fordit tesztjéhez ez indítandó
Sub Teszt1()
'Híváskor a cím szerinti (ByRef) paraméter (st) helyén állhat érték is
MsgBox Fordit("Kitűnő vőt rokonok orrtövön ütik")
End Sub

'Sztring megfordítása eljárással
Sub MegFordit(st As String)
Dim i As Integer, er As String
er = ""
For i = 1 To Len(st)
  er = Mid(st, i, 1) + er
Next
'A cím szerinti (ByRef) paraméterben (st) adjuk vissza az eredményt
st = er
End Sub

'A MegFordit tesztjéhez ez indítandó
Sub Teszt2()
Dim s As String
'Call esetén kell a zárójel
s = "Indul a görög aludni": Call MegFordit(s): MsgBox s
'Call nélkül nem kell a zárójel
s = "Géza kék az ég": MegFordit s: MsgBox s
End Sub
```

A fejezet lezárásaként megjegyezzük, hogy a VB nem engedi meg a szubrutinok egymáson belüli deklarálását, de a rekurziót (amikor is egy szubrutin önmagát hívja meg, vagy több szubrutin hívja egymást kölcsönösen) igen. A rekurzív algoritmusok iránt érdeklődőknek a szakirodalmat (pl. [3], [4]) ajánljuk.

2.5. Összetett adattípusok

Az eddig megismert adattípusokkal ellentétben, ahol egy változóban csak egy adatot tárolhatunk egy időben, az összetett adattípusok segítségével több adat együttes kezelésére is lehetőségünk nyílik.

2.5.1. Tömbök

A tömb egy általánosan és gyakran használt eszköz a szoftverfejlesztők, programozók körében, mivel a tömbökkel kényelmesen kezelhetünk több, azonos típusú adatot. Az adatok elérésére, vagyis a tömbök elemeinek kiválasztására sorszámokat, más néven indexeket használunk.

Ha az elemeket egy sorszámmal azonosítjuk, akkor egydimenziós tömbről, ha két sorszámmal, akkor kétdimenziós tömbről, ha *n* db sorszámmal azonosítjuk, akkor *n* dimenziós tömbről beszélünk. Egy tömb egy elemének kiválasztásához tehát pontosan annyi index szükséges, ahány dimenziós az adott tömb.

Ahogy az egyszerű adattípusok esetén is megkülönböztettük az adatot (pl. 3) az őt tároló változótól (pl. `i`) és annak típusától (pl. `Integer`), itt is kitérünk ezek különbözőségére. A tömb elnevezés ugyanis a rövidsége miatt mind a három esetben használatos, így a tömb lehet:

- *Tömbadat:* amely egydimenziós tömb esetén egy adatsornak, vagy matematikai fogalommal egy vektornak, kétdimenziós tömb esetén egy adattáblázatnak, vagy mátrixnak felel meg.
- *Tömb adattípus:* amely definiálja a tömb dimenzióit és a tömb elemeinek típusát.
- *Tömbváltozó:* amelyben az adatok tárolódnak. Egy elem kiválasztásával, azaz a tömbváltozó indexelésével egy, a tömb elemtípusával megegyező típusú változót hivatkozunk.

Egy tömböt a használata előtt deklarálni kell, azaz definiálnunk kell a tömb elemeinek típusát és a tömb dimenzióit.

A deklarálás (egyszerűsített) szintaktikája:

```
Dim varname[([subscripts])] [As type] [,...]
```

Az indexek (*subscripts*) megadásának szintaktikája:

```
[lower To] upper [,[lower To] upper] ,...
```

Pl.

```
Dim a(10) As Integer                'Egydimenziós tömb
Dim b(1 To 5, -3 To 2) As Double    'Kétdimenziós tömb
```

Vegyük észre, hogy az egyes dimenziók alsó indexhatára elhagyható. Az explicit módon nem definiált alsó indexhatárok értékeit az `Option Base` (modulszintű) utasítás határozza meg.

Az utasítás szintaktikája:

```
Option Base {0|1}
```

Értelemszerűen a megadott érték (0 vagy 1) lesz a tömbök alsó indexhatára. Alapértelmezésben az `Option Base 0` van érvényben, azaz a tömbelemek alsó indexhatára 0.

Megjegyzés: Noha a VB-ben egy tömb dimenzióinak maximális száma 60 lehet, azaz használhatunk három, négy, …, 60 dimenziós tömböket, a gyakorlatban az egy- és kétdimenziós tömbök a leggyakoribbak.

Az előzőekben ismertetett deklarálás ún. *statikus* tömböket deklarál, amelyek mérete fordítási időben meghatározódik, és futási időben már nem változtatható. A VB azonban megengedi az ún. *dinamikus* tömbök használatát is, amelyek mérete a program futása során megváltoztatható. Ezeket a tömböket is a `Dim` utasítással deklaráljuk, de az első deklarálásnál csak a tömbelemek típusát adjuk meg, a dimenziókat nem. A tömb átméretezése ezután a `ReDim` utasítással történik, amelyben előírhatjuk, hogy a már definiált értékű tömbelemek értékeit megőrizzük-e (`Preserve`) vagy sem.

Ha egy tömb méretét kisebbre állítjuk, akkor az „eltűnt” tömbelemek értékeit elveszítjük, míg egy tömb méretének növelésekor „keletkező” új elemek (a deklarált változókhoz hasonlóan) numerikus elemtípusú tömb esetén 0, a logikai elemek esetén `False`, míg `String` típus esetén az üres sztring kezdőértéket kapják. Ilyen értékeket kapnak az új tömb elemei akkor is, ha a `Preserve` kulcsszót elhagyjuk.

Szintaktika:

```
ReDim [Preserve] varname(subscripts) [As type] [,...]
```

Pl.

```
Dim a() As Integer      'Dinamikus tömb egészekből
ReDim a(5)              'A tömb legyen 5 elemű
For i = 1 To 5          'A tömb használata
  a(i) = i
Next
ReDim Preserve a(10)    'A tömb legyen 10 elemű
                        'Az első 5 elem értéke megmarad, a többi 0 lesz
```

Megjegyzés

- A példában feltettük, hogy a modulban az `Option Base 1` utasítás van érvényben.
- Egy dinamikus tömb többször is átméretezhető, de az elemek típusa nem változtatható meg.
- A dinamikus tömbök a `ReDim` utasítással közvetlenül (a `Dim` utasítás nélkül) is deklarálhatók, ekkor az elemek típusát is itt adjuk meg (pl. `ReDim a(5) As Integer`).

Az, hogy statikus vagy dinamikus tömböket használunk-e egy feladat megoldása során, az attól függ, hogy ismerjük-e előre az adatok maximális számát vagy sem. Ha ismerjük, akkor statikus (az adott maximumra deklarált) tömböt használunk, és az adatok aktuális számát egy külön változóban tároljuk. Ebben az esetben csak akkor lesz helypazarló a megoldásunk, ha az adatok maximális száma (azaz a tömb által lefoglalt memória mérete) jóval nagyobb, mint az adatok aktuális száma.

Ha az adatok maximális számát nem tudjuk előre meghatározni, akkor a megoldást dinamikus tömbökkel végezzük. Ebben az esetben a tömböt mindig akkorára deklaráljuk, hogy az adataink éppen elférjenek benne, így a megoldás sose foglal le több memóriát, mint amennyi szükséges.

A VB az `UBound` (felső határ) és `LBound` (alsó határ) függvényekkel biztosítja a tömbök deklarált indexhatárainak lekérdezhetőségét.

Szintaktika:

```
UBound(arrayname[,dimension])
LBound(arrayname[,dimension])
```

arrayname	A tömb azonosítója.
dimension	A lekérdezendő dimenzió sorszáma (alapértelmezett értéke 1).

Pl.

```
Dim a(10) As Integer
Dim b(1 To 5, -3 To 2) As Double

UBound(a) → 10   UBound(b) → 5    UBound(b,2) → 2
LBound(a) → 0    LBound(b) → 1    LBound(b,2) → -3
```

Megjegyzés: A példában feltettük, hogy a modulban az `Option Base 0` utasítás érvényes.

2.5.2. Rekordok

A tömbök (legyenek azok egy-, vagy többdimenziósak) *azonos típusú* adatok egy összességének használatát biztosították. Ha azonban *különböző típusú* adatok egy összességét szeretnénk együtt kezelni, akkor egy újabb típust, a rekord adattípust kell használnunk. A rekord adattípussal ugyanis több, tetszőlegeses számú és típusú adatot foglalhatunk egy logikai egységbe.

Ezt az adattípust külön kell definiálnunk, ugyanis meg kell adnunk a rekord felépítését, azaz a rekordot alkotó *mezők* azonosítóját és típusát.

A rekordtípus megadásának (egyszerűsített) szintaktikája:

```
Type name
  elementname[([subscripts])] As type
  elementname[([subscripts])] As type
  ...
End Type
```

name	A rekordtípus azonosítója.
elementname	Az adott mező azonosítója.
subscripts	Ha a mező tömb, akkor annak indexhatárai.
type	Az adott mező típusa.

Pl.

```
Type Adat
  Nev As String * 20          'Fix (20) hosszú sztring
  Jegyek(1 To 2) As Byte
End Type
```

A példa egy olyan rekordtípust (`Adat`) definiál, amelynek két mezője van. Az első mező (`Nev`) egy fix hosszú sztring, a második mező (`Jegyek`) egy 2 elemű, egydimenziós, `Byte` elemtípusú tömb.

Megjegyzés

- Az egyes mezők beljebb tagolása nem kötelező, csak az áttekinthetőséget segíti.
- A példában fix hosszú sztringtípust (`String * 20`) használtunk, de a változó hosszú `String` típust is használhattuk volna.
- A `Type` utasítás modulszintű utasítás.

Egy definiált rekordtípus ugyanúgy használható változók deklarálására, mint a VB többi adattípusa.

Pl.

```
Dim h As Adat, a(1 To 3) As Adat
```

A példa két változót deklarál. A `h` változóban egy rekord adatai tárolhatók, míg az `a` változó három rekord tárolására alkalmas.

A rekord mezőire való hivatkozás:

```
rekordváltozó.mezőnév
```

A rekordok egyes mezőire úgy hivatkozhatunk, hogy a rekordváltozó és a mezőnév közé egy pontot teszünk. Ezzel a hivatkozással egy, a mező típusával megegyező típusú változót hivatkozunk, amit ennek megfelelően használhatunk. A rekordokra nem definiáltak műveletek, de az azonos típusú rekordváltozók között megengedett az értékadás (lásd 2.5.3. fejezet).

Pl.

```
h.Nev = "Halász Helga"
a(1).Jegyek(1) = 5
```

A mezőkre való hivatkozás a `With` utasítás segítségével rövidíthető. Ekkor a rekordváltozót elegendő egyszer megadni (a `With` kulcsszó után), a mezőnevek előtt elhagyhatjuk, de a pontot ki kell tennünk.

Pl.

```
With a(2)
  .Nev = "Vadász Viktória": .Jegyek(1) = 3: .Jegyek(2) = 4
End With
```

A példa a korábban deklarált a tömb második elemében tárolt rekord egyes mezőinek ad értéket.

Megjegyzés

- A `With` utasítás objektumokra is használható. Az objektumokról a 2.6., illetve 4.1. fejezetekben lesz szó.
- A `With` blokkban szereplő utasítások beljebb tagolása nem kötelező, csak az áttekinthetőséget segíti.
- A `With` utasítások egymásba ágyazhatók.
- A rekord szó a szakirodalomban általánosan használt (ezért a dokumentumban is ezt használtuk), de a VB súgója a rekord adattípusra a *felhasználó által definiált adattípus* (user-defined data type) elnevezést használja.
- Az összetett adattípusú (tömb, rekord) deklarált változók is kapnak kezdőértéket, amit a tömbelemek, illetve az egyes mezők típusa határoz meg (lásd 2.2.1. fejezet).

2.5.3. Mintafeladat

A tömbök és rekordok használatát az alábbi feladat segítségével szemléltetjük.

Feladat: Egy hallgatói nyilvántartásban ismerjük a hallgatók nevét és adott számú érdemjegyét. Minden hallgatónak ugyanannyi érdemjegye van. Az adatokat egy rekordokból álló egydimenziós tömbben tároljuk. Készítsünk névsor szerint rendezett listát, amelyen a hallgatók érdemjegyei is szerepelnek!

Az alábbi megoldásban először egy konstanst deklarálunk, amely definiálja a hallgatók érdemjegyeinek számát. Az olyan értékeket, amelyek a programban több helyen is szerepelnek, célszerű névvel ellátni, azaz külön deklarálni, mert akkor az érték esetleges módosításához elegendő egy helyen (a deklarációban) módosítanunk.

A konstansok deklarálásának szintaktikája:

```
[Public|Private] Const constname [As type] = expression
```

`Public`	A konstans hivatkozható lesz minden modulból.
`Private`	A konstans csak abban a modulban hivatkozható, ahol deklarálták (ez az alapértelmezés).
`constname`	A konstans azonosítója.
`type`	A konstans típusa (opcionális).
`expression`	A konstans értékét megadó kifejezés.

Pl.

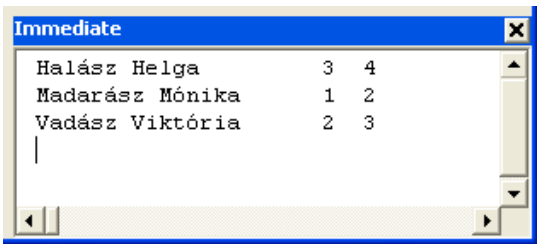
```
Const db = 2
```

Megjegyzés

- Az utasítás modul szinten is, és szubrutinon belül is használható. Az utóbbi esetben a konstans csak az adott szubrutinon belül hivatkozható, és a láthatóságát (hivatkozhatóságát) nem lehet a `Public` és `Private` kulcsszavakkal módosítani.
- A konstans értékét megadó kifejezésben konstans értékek, deklarált (névvel azonosított) konstansok, valamint műveletek állhatnak.
- A típust (*type*) általában elhagyjuk (a példában is így tettünk), ekkor a konstans típusa a kifejezés értékének legmegfelelőbb típus lesz (esetünkben `Byte`).

A konstans definiálása modul szinten történt, hiszen az egész modulban szeretnénk használni. A rekordtípus (amely leírja egy hallgató adatait) deklarációja viszont csak modul szinten (azaz a modul elején) történhet. A nevekhez fix hosszú sztring típust használtunk, amely nem engedi meg, hogy a nevek a megadott hossznál (a példában 20) hosszabbak legyenek.

Az adatok kiírását egy szubrutin (`Kiiras`) végzi, amely egy `Adat` rekordokból álló tömböt kap paraméterként (`a`), amit a paraméter után szereplő (üres) zárójelpár jelez. A kapott tömböt egydimenziós tömbként kezeljük, és feltesszük, hogy a tömb minden eleme definiált, amit a `Kiiras` szubrutin hívója (`RendezesTeszt`) biztosít, ezért a tömbelemeken végiglépdelő ciklus szervezését a tömb lekérdezett (`LBound`, `UBound`) indexhatárai alapján végezhetjük. Az adatkiírások után akkor emelünk sort (egy paraméter nélküli `Debug.Print` hívással), ha az adott hallgató összes adatát (név, érdemjegyek) kiírtuk, így minden hallgató külön sorba kerül. Az eredménylista a 2.3. ábrán látható. Vegyük észre a nevek jobbról szóközökkel való feltöltöttségét (ez a fix hosszú sztringek értékadásakor automatikusan megtörténik).



2.3. ábra. A mintafeladat eredménylistája

Az adatok rendezését a `Rendezes` szubrutin végzi, amely (a `Kiiras` szubrutinhoz hasonlóan) szintén megkapja a rendezendő adatok tömbjét. Az adatkiírás nem változtatott a paraméterként kapott adatokon, de a rendező szubrutinnak éppen ez a dolga, nevezetesen hogy névsor szerinti sorrendbe tegye a tömbben lévő adatokat. Mivel a tömb átadására deklarált (`a`) paraméter cím szerinti átadású (`ByRef` az alapértelmezés, és tömböket nem is adhatunk át érték szerint, lásd 2.4.1. fejezet), ezért a tömb elemein végzett változtatásokat a hívó is „érzékelni fogja”.

A rendezést egy, a minimum kiválasztás elvén alapuló rendező algoritmus segítségével valósítottuk meg. A külső ciklus minden egyes lépésében megkeressük (a belső ciklussal) a még rendezetlen tömbrész legkisebb elemét (annak indexét tárolja a `k` változó), és ha ez az elem nincs a helyén (az `i`-edik helyen), akkor odatesszük. A rekordok között értelmezett az értékadás, ezért két elem (`i`-edik és `k`-adik) felcserélése elvégezhető három értékadással (nevezetesen megjegyezzük az egyiket, oda áttesszük a másikat, majd a másikba betesszük a megjegyzett egyiket). Vegyük észre, hogy a cseréhez használt változó (`cs`) a tömbelemekkel megegyező (`Adat`) típusú.

A minimumkeresés úgy történik, hogy megjegyezzük az adott tömbrész első elemének indexét, és megvizsgáljuk a többi elemet. Ha kisebbet találunk (a rekordok `Nev` mezője szerinti összehasonlítás szerint, hiszen névsor szerinti sorrendet szeretnénk előállítani), akkor annak indexét jegyezzük meg (hiszen most már ő az aktuálisan legkisebb). A rendezetlen tömbrész kezdetben (`i=1` esetén) az egész tömb, utána (`i=2` esetén) csak a második elemtől az utolsó elemig (hiszen az első, vagyis a legkisebb elem már a helyére, a tömb elejére került), és így tovább, legvégül (`i=UBound(a)-1` esetén) már csak a tömb két utolsó elemét tartalmazza.

A `Rendezes` szubrutint a `RendezesTeszt` szubrutin hívja meg azután, hogy kezdőértékekkel látta el az adattárolásra használt tömbváltozót. A tömb azonosítására mindkét helyen ugyanazt az azonosítót (`a`) használtuk, de használhattunk volna különböző azonosítókat is. A harmadik személy (akinek adatait a

tömb második elemében tároljuk, hiszen a modulban alapértelmezésben az `Option Base 0` érvényes) adatainak megadásakor a rekordmezőkre a `With` utasítás segítségével hivatkoztunk, hogy ezt is bemutassuk.

Megjegyzés

- A megoldás csak a VB nyelv lehetőségeit használta, így szükség volt egy rendező algoritmus programozására. Ha azonban kihasználtuk volna az Excel VBA lehetőségeit, akkor a hallgatók adatait egy Excel munkalapon tárolva egyetlen metódushívással „elintézhető" lett volna az adatok rendezése. Az Excel objektumairól és használatukról a 4.1. fejezetben lesz szó.
- Igen sokféle rendező algoritmus ismert, amelyekből ez egyik legegyszerűbbet választottuk ki. A témakör iránt érdeklődőknek a szakirodalmat (pl. [3], [4]) ajánljuk.

```
'Egy hallgató érdemjegyeinek száma
Const db = 2

'Egy hallgató adatait leíró rekordtípus
Type Adat
  Nev As String * 20          'Fix (20) hosszú sztring
  Jegyek(1 To db) As Byte
End Type

'Adat típusú elemekből álló tömb kiírása
Sub Kiiras(a() As Adat)
Dim i As Integer, j As Integer
For i = LBound(a) To UBound(a)
  Debug.Print a(i).Nev;
  For j = 1 To db
    Debug.Print a(i).Jegyek(j);
  Next
  Debug.Print
Next
End Sub

'Adat típusú elemekből álló tömb rendezése név szerint
Sub Rendezes(a() As Adat)
Dim cs As Adat, i As Integer, j As Integer, k As Integer
For i = LBound(a) To UBound(a) - 1
  k = i
  For j = i + 1 To UBound(a)
    If a(j).Nev < a(k).Nev Then
      k = j
    End If
  Next
  If k > i Then
    cs = a(i): a(i) = a(k): a(k) = cs
  End If
Next
End Sub
```

```
'A rendezés tesztjéhez ez indítandó
Sub RendezesTeszt()
Dim a(2) As Adat
a(0).Nev = "Madarász Mónika": a(0).Jegyek(1) = 1: a(0).Jegyek(2) = 2
a(1).Nev = "Vadász Viktória": a(1).Jegyek(1) = 2: a(1).Jegyek(2) = 3
With a(2)
  .Nev = "Halász Helga": .Jegyek(1) = 3: .Jegyek(2) = 4
End With
Call Rendezes(a)
Call Kiiras(a)
End Sub
```

2.6. Objektumok, objektumtípusok

Az objektum szót már használtuk (pl. a Debug objektum kapcsán), de még nem beszéltünk arról, hogy pontosan mit jelent, mit takar az objektum elnevezés az informatikában.

Objektumon adatok és a rajtuk értelmezett tevékenységek egy logikailag összefüggő egységét értjük. Az objektum (hasonlóan, mint a tömb) egy gyakran és általánosan használt szó, ami jelenthet adatot (egy konkrét objektumpéldányt), változót (ami tárolja és elérhetővé teszi az objektum adatait és tevékenységeit), valamint típust is (az objektum típusát, amely definiálja az objektum adatainak típusát és az objektum tevékenységeit).

Az objektumok tevékenységeit definiáló szubrutinokat *metódusoknak* (methods), az objektumok típusát *osztályoknak* (class) nevezzük. Az objektumok adataihoz általában *tulajdonságok* (properties) segítségével férhetünk hozzá, amelyek védik az adatok helyességét (lásd 3.2.3. fejezet).

Az objektumok azon metódusait, amelyek valamilyen *esemény* bekövetkezésekor (pl. egy nyomógomb esetén a nyomógomb feletti kattintás) aktivizálódnak, *eseménykezelőknek* nevezzük. Az események általában valamilyen felhasználói interakció eredményeként következnek be, de más futó folyamatok is kiválthatják az egyes események bekövetkezését (pl. meghívunk egy eseménykezelőt).

Megjegyzés: A VBA megengedi az osztályok definiálását, de mi csak a „készen kapott” osztályokat fogjuk használni.

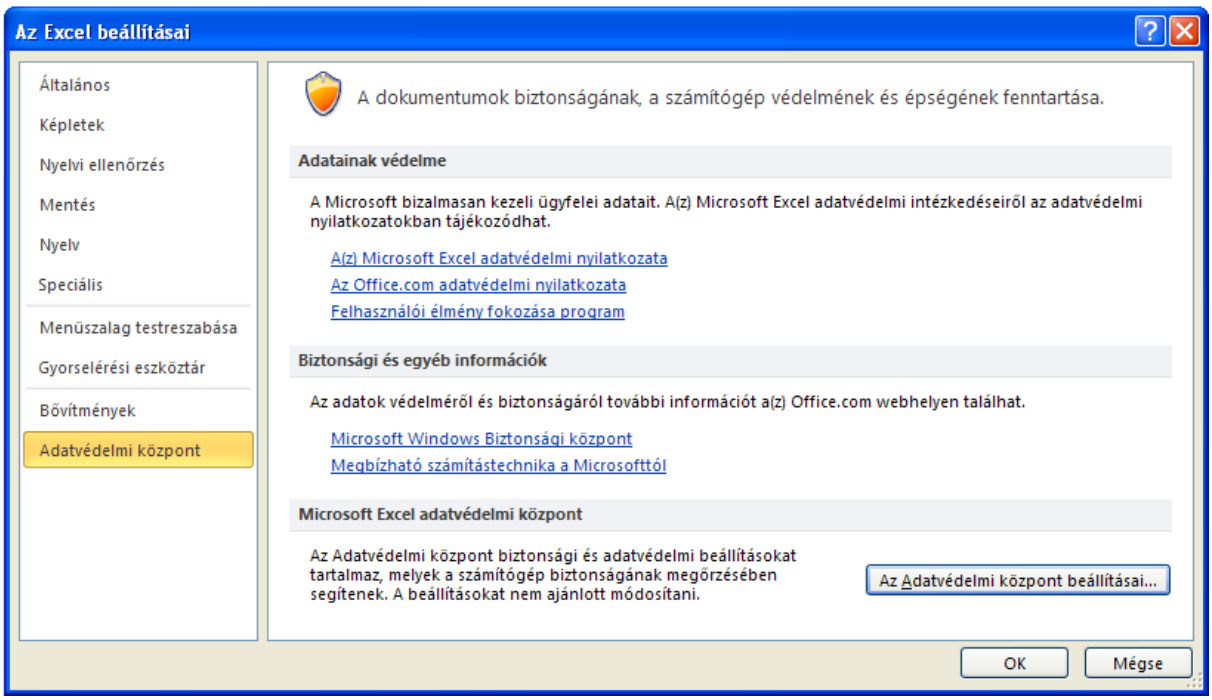
3. AZ EXCEL VBA HASZNÁLATA

Ebben a fejezetben az Excel VBA fejlesztőkörnyezetről lesz szó. Elsősorban a forrásprogramok megadásához, azok futtatásához, a hibakereséshez kapcsolódó funkciókat részletezzük, de itt kapott helyet a vizuális formtervezés is.

3.1. Makrók

„A makró parancsok sorozata, amely ismétlődő feladatok végrehajtásának automatizálására használható.” – idézet az Excel súgójából. A makró szó az adott MS Office (esetünkben az MS Excel) alkalmazás programfejlesztő környezetében létrehozott objektumok és VBA forráskódok általános, összefoglaló elnevezése. Az Excel 2010 a makrókat tartalmazó dokumentumokra a „Makróbarát Excel munkafüzet” elnevezést használja. A makrókat tartalmazó dokumentumokat ilyen fájlként mentsük (ezek kiterjesztése xlsm).

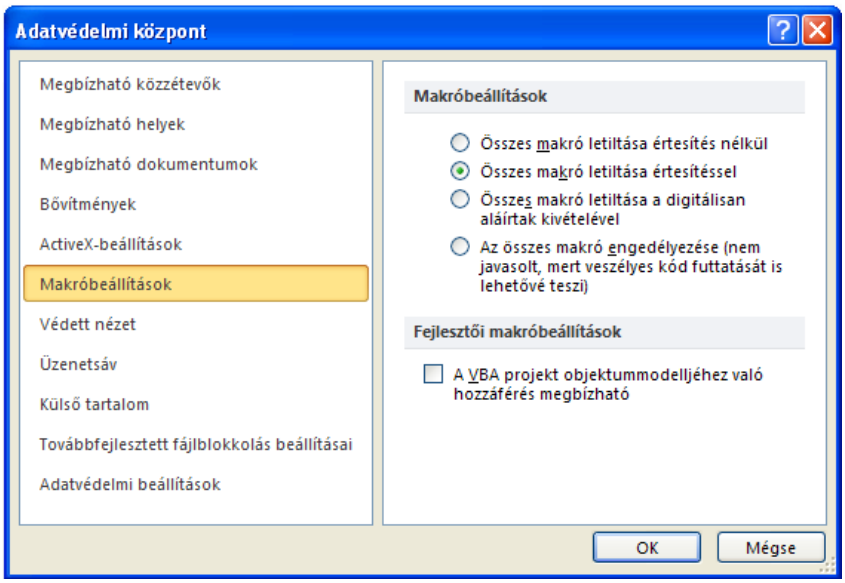
A makróvírusok elterjedésének megakadályozása érdekében az Excel bizonyos beállításai a makrókat tartalmazó dokumentumok kezelésére vonatkoznak. A Fájl, Beállítások, Adatvédelmi központ (lásd 3.1. ábra) ablakban a jobb alsó sarokban található „Az Adatvédelmi központ beállításai…” nyomógomb segítségével (többek között) megadhatók a makrókra vonatkozó beállítások is (lásd 3.2. ábra).



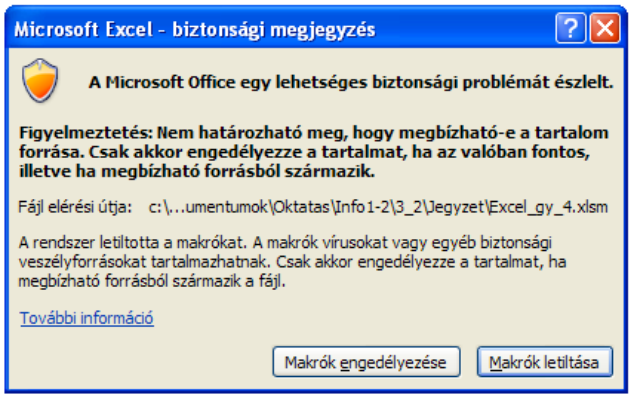
3.1. ábra. Az adatvédelmi központ

Ahhoz, hogy a makrókat tartalmazó dokumentumok megnyitásakor eldönthessük, hogy engedélyezzük vagy letiltjuk az adott dokumentumban lévő makrókat, az „Összes makró letiltása értesítéssel” választógombot kell bekapcsolni (lásd 3.2. ábra). Ebben az esetben egy makrókat tartalmazó dokumentum megnyitásakor üzenetet kapunk. Ha a Visual Basic Editor meg van nyitva, akkor a 3.3. ábrán szereplő párbeszédablak jelenik meg, egyébként meg a 3.4. ábrán látható üzenetsáv. Mindkét esetben engedélyezhetjük a dokumentumban lévő makrókat.

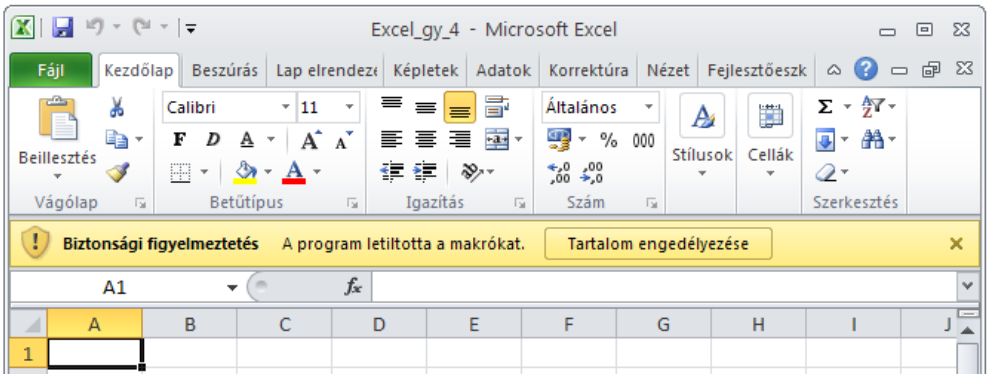
Az engedélyezés, illetve letiltás csak a makrók futtatására vonatkozik, így letiltott esetben is megnézhetjük, sőt akár módosíthatjuk is a makrókat. Az engedélyezés, illetve tiltás (adott dokumentumra vonatkozó) megváltoztatásához zárjuk be és nyissuk meg újra (a kívánt módon) az adott dokumentumot.



3.2. ábra. Az adatvédelmi központ makróbeállításai



3.3. ábra. Makrók engedélyezése, illetve letiltása párbeszédablak



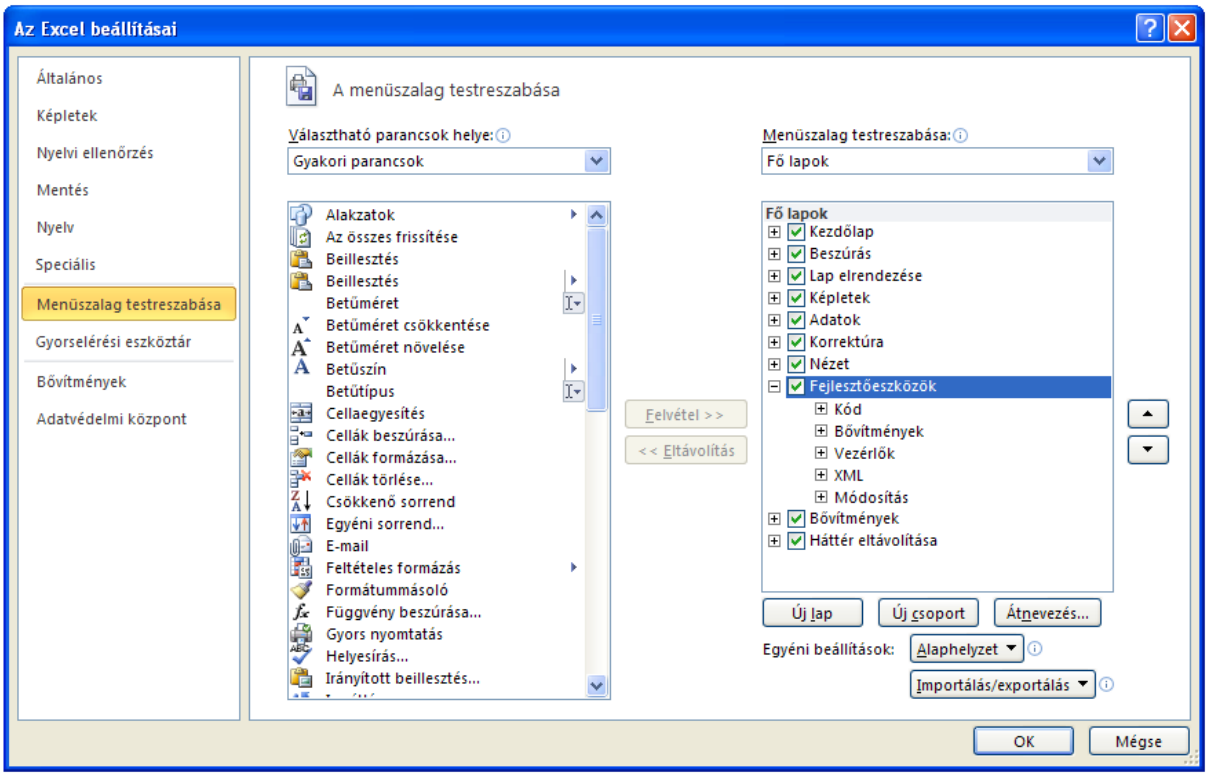
3.4. ábra. Makrók engedélyezése, illetve letiltása üzenetsáv

Megjegyzés

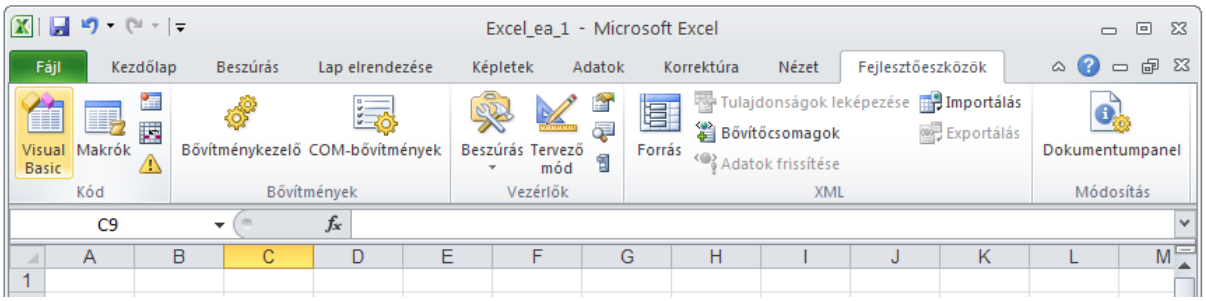
- Ha a 2.7. ábrán látható sárga színű üzenetsávban engedélyezzük a makrókat („Tartalom engedélyezése” gomb), akkor az adott számítógépen a dokumentum megbízható dokumentumként kezelődik a továbbiakban, ezért újabb megnyitásakor már nem kell külön engedélyezni a makrókat, egyébként (az üzenetsáv bezárásával) meghagyjuk a makrók letiltását.
- Egy újonnan létrehozott munkafüzetben készített makrók a beállításoktól függetlenül mindig futtathatók.

3.2. A Visual Basic Editor

Alapértelmezésben az Excel menüszalagján nem látható a Fejlesztőeszközök lap (lásd 3.6. ábra), amelyen a Visual Basic fejlesztőrendszer, a Visual Basic Editor elérhető, de a gyorsbillentyűjével (Alt+F11) ekkor is megnyitható. A Fejlesztőeszközök lap megjelenítéséhez a Fájl, Beállítások, Menüszalag testreszabása párbeszédablakban be kell kapcsolni a lap megjelenítését szabályozó jelölőnégyzetet (lásd 3.5. ábra).

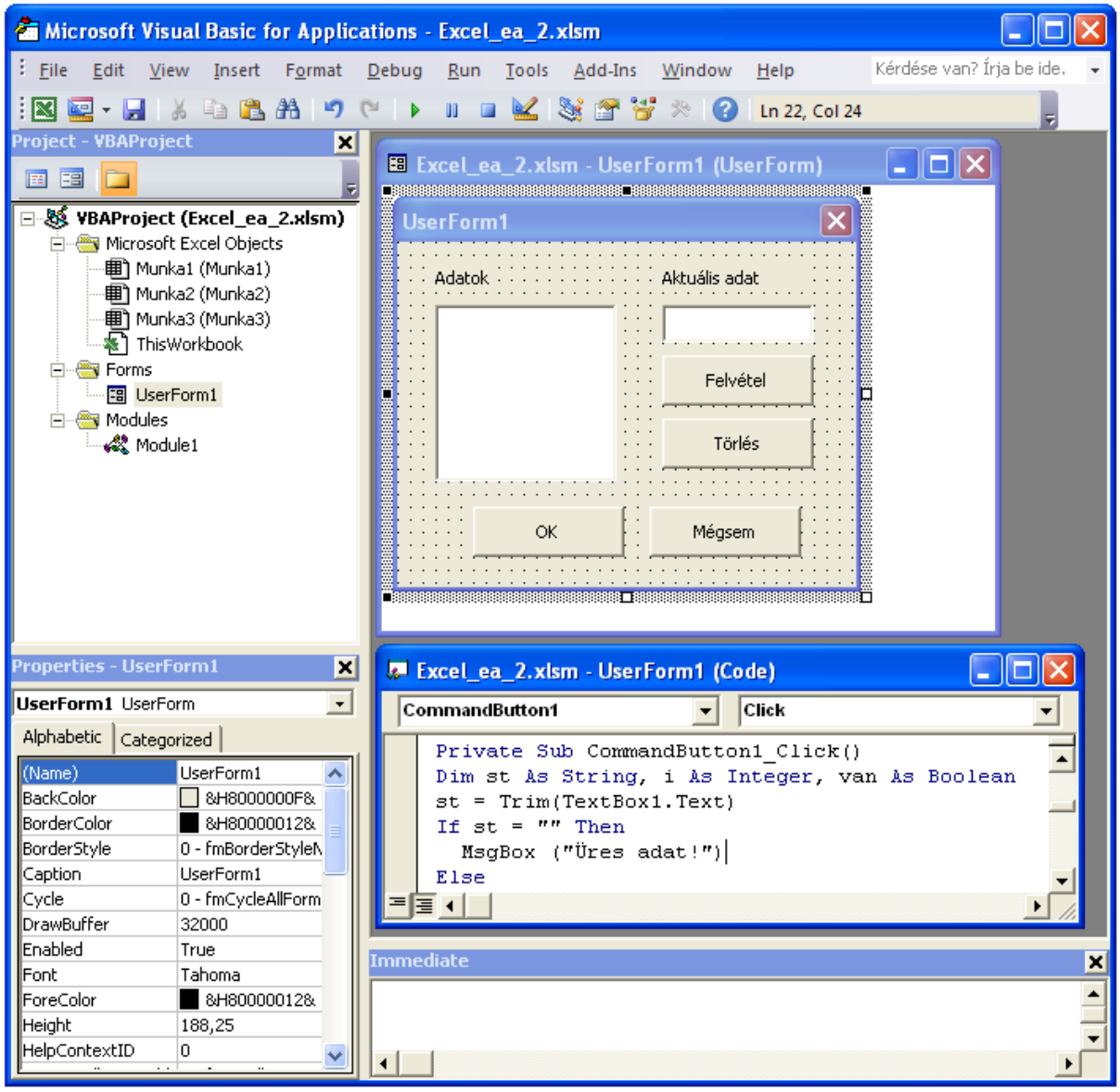


3.5. ábra. A menüszalag testreszabása



3.6. ábra. A menüszalag Fejlesztőeszközök lapja

A Visual Basic Editor egy önálló ablakban jelenik meg, de az Excel bezárásával ez az ablak is bezáródik. A fejlesztőkörnyezet angol nyelvű (menü, súgó, hibaüzenetek, stb.), hiába magyar nyelvű az Excel, amihez tartozik.



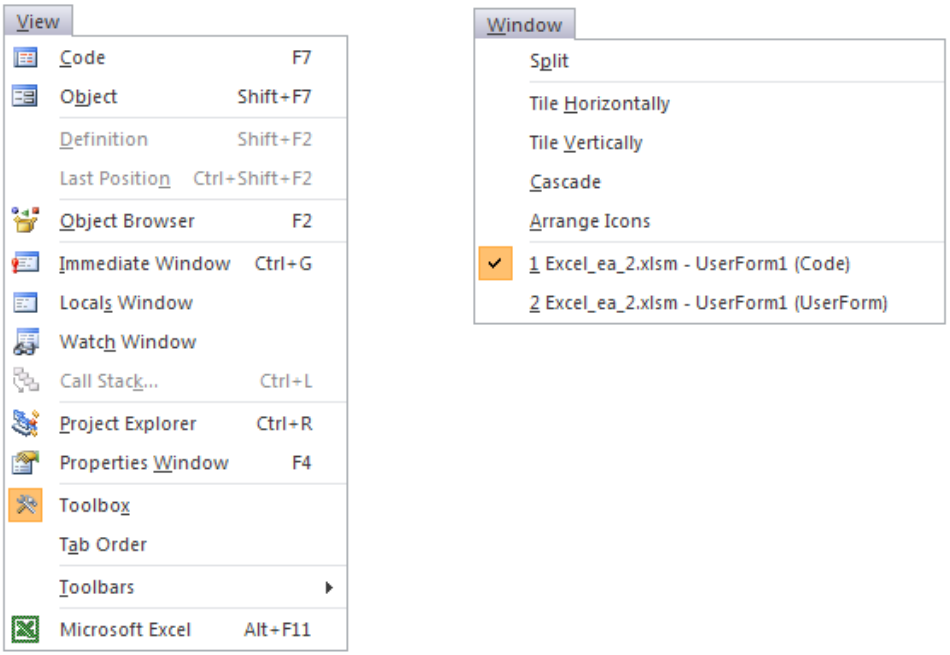
3.7. ábra. A Visual Basic Editor

Bár az ablak elrendezése beállítható, egy tipikus elrendezés a 3.7. ábrán látható. Felül a menüsor, alatta az eszköztársor, balra fent a Project ablak, alatta a Properties ablak, a jobb oldali, szürke hátterű munkaterületen most két ablak látható. Az egyik a UserForm1 objektum tervezőablaka, a másik a UserForm1 objektumhoz tartozó forráskód szerkesztőablaka. A munkaterület alatt az Immediate ablak látható.

Valamennyi ablak szabadon megjeleníthető, méretezhető, más helyre tehető, vagy akár be is zárható. A View menü segítségével (lásd 3.8. ábra) bármikor újra megjeleníthető egy bezárt ablak. A gyakran használatos funkciók a helyi menü segítségével, vagy a hozzájuk tartozó gyorsbillentyűkkel is aktivizálhatók (pl. F7, Shift+F7). A helyi menü (a Windows-ban megszokott módon) az adott helyen, az egér jobb gombjával hozható be.

Megjegyzés

- A Visual Basic Editor-ban létrehozott makrókat nem kell külön menteni, ezek a munkafüzet mentésekor elmentődnek. A Visual Basic Editor mentés (Save) funkciója az egész munkafüzetet menti (csakúgy, mint az Excel-beli mentés funkció).
- Bizonyos ablakok (pl. Project, Properties, Immediate) rögzíthetők (dockable) a munkaterületen, sőt akár a Visual Basic Editor ablakán kívülre is mozgathatók. Az ablakelrendezés (mely ablakok hol és mekkora méretben jelennek meg) megőrződik, minden belépéskor a legutóbbi ablakelrendezésben jelenik meg a Visual Basic Editor.
- A nem rögzíthető ablakok a nyitáson és záráson kívül még minimalizálhatók (amikor is ikonként a munkaterület aljára kerülnek) és maximalizálhatók (ilyenkor a teljes munkaterületet kitöltik eltakarva egymást). A nyitott ablakok közötti váltás a kívánt ablakon való kattintás mellett, a Window menü segítségével is elvégezhető, amely a Windows alkalmazásokban szokásos funkciókkal (ablakok különféle elrendezése, a nyitott ablakok listája, stb.) rendelkezik (lásd 3.8. ábra).



3.8. ábra. A Visual Basic Editor View és Window menüje

A fontosabb ablakok szerepéről a következő alfejezetekben lesz szó.

3.2.1. Az Immediate ablak

Az Immediate ablak amellett, hogy a programok egyik output megjelenítő eszköze (lásd 2.2.6. fejezet), utasítások közvetlen végrehajtására is használható. A végrehajtani kívánt utasításokat egy sorba kell írni, több utasítás esetén az utasítások közé kettőspontot kell tenni (lásd 2.3.1. fejezet). A végrehajtást az Enter billentyű leütésével kérhetjük.

Az Immediate ablak tartalma szabadon szerkeszthető, így pl. a korábban végrehajtott sorok (esetleges módosítással) újra végrehajthatók, a felesleges sorok törölhetők, stb.

Pl.

```
i=2:j=3
Print i;"*";j;"=";i*j;tab(1);i;"^";j;"=";i^j
?"1-10-ig az egész számok":for i=1 to 10:?i;:next:?
```

Megjegyzés

- A Print és a ? a Debug objektum Print metódusát (Debug.Print) hivatkozza röviden.
- Nem minden VB utasítás hajtható végre az Immediate ablakban (pl. `Dim` utasítás).
- Természetesen itt is csak szintaktikailag helyes utasítások hajthatók végre. Az esetleges hibaüzenetet az Enter billentyű leütése után kapjuk meg (szemben a kódszerkesztő ablakban beírt utasításokkal, ahol már a szintaktikailag hibás sor elhagyásakor megkapjuk a megfelelő hibaüzenetet, ha az erre vonatkozó kapcsoló (Auto Syntax Check, lásd 3.2.4. fejezet 3.20. ábra) bekapcsolt állapotú.
- Az Immediate ablakban csak akkor lehet utasításokat végrehajtani, ha a munkafüzetre vonatkozó biztonsági beállítások engedélyezik a makrók futtatását (lásd 3.1. fejezet).

3.2.2. A Project ablak

Az MS Office VBA fejlesztőkörnyezetben a projektek foglalják egységbe az adott dokumentumot a hozzá tartozó makrókkal. A projekt az adott dokumentumfájlban tárolt objektumok áttekinthetőségét, kezelését hivatott szolgálni. A Project ablak a Visual Basic Editor View menüjének Project Explorer funkciójával (lásd 3.8. ábra) jeleníthető meg (lásd 3.7. ábra).

Egy Excel projektben az alábbi objektumok szerepelnek, illetve szerepelhetnek:

- Excel objektumok (Microsoft Excel Objects)
  - Az egyes munkalapok (Munka1,…)
  - A munkafüzet (ThisWorkbook)
- Formok (Forms)
- Modulok (Modules)
- Osztálymodulok (Class Modules)

Az Excel objektumok csoport elemei az adott munkafüzethez igazodnak (pl. annyi munkalap objektum szerepel a projektben, ahány munkalapja van az adott munkafüzetnek). A munkalap objektumokkal az egyes munkalapok, a ThisWorkbook objektummal a (makrókat tartalmazó) munkafüzet hivatkozható. Ezek az objektumok nem hozhatók létre, illetve nem törölhetők a Visual Basic Editor-ban, de a hozzájuk tartozó esetleges eseménykezelők (pl. a munkafüzet megnyitása, bezárása, egy munkalap aktivizálása, stb.) az objektumhoz tartozó modulban definiálhatók (lásd 4.2.4 fejezet).

A többi csoport (formok, modulok, osztálymodulok) elemeit szabadabban kezelhetjük. Létrehozhatunk (Insert), törölhetünk (Remove), behozhatunk (Import), kivihetünk (Export) elemeket (lásd 3.10. ábra).

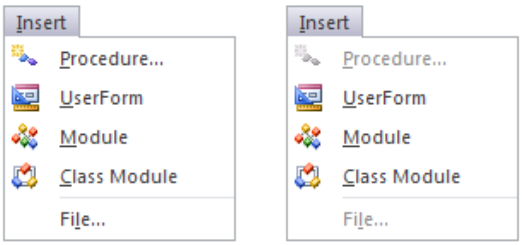
A formokról, a vizuális formtervezésről a 3.2.7. fejezetben lesz szó.

A modulok csoport moduljai a forráskódok megadására használatosak. A logikailag összetartozó forráskódokat célszerű egy modulba tenni. Az eseménykezelőket a megfelelő objektum moduljában (azaz nem a modulok csoportban) kell elhelyezni. A modulok felépítéséről a 3.2.5. fejezetben lesz szó.

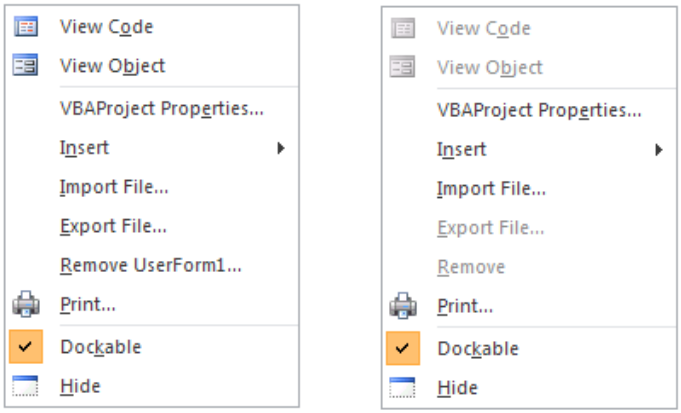
Az osztálymodulokban osztályokat (objektumtípusokat) definiálhatunk.

Az objektumokhoz tartozó modulok megnyitása (a Project ablak megfelelő elemének kiválasztása után) a View menü Code funkciójával (lásd 3.8. ábra), vagy a Project ablak helyi menüjének View Code funkciójával (lásd 3.10. ábra) történhet. A modulok csoport moduljait még (a megfelelő modulon való) dupla kattintással is megnyithatjuk.

Új objektum létrehozása a Visual Basic Editor Insert menüjével (lásd 3.9. ábra), vagy a Project ablak helyi menüjével (lásd 3.10. ábra) történhet. A menükben található menüpontok választhatósága az aktuális állapottól (kódszerkesztésben voltunk-e éppen vagy sem), illetve a Project ablak aktuális elemétől függ.



3.9. ábra. A Visual Basic Editor Insert menüje



3.10. ábra. A Project ablak helyi menüje

A Project ablak elemeit a helyi menü Remove funkciójával törölhetjük, ahol a menüpont felirata a kiválasztott objektum nevét is tartalmazza (lásd 3.10. ábra bal oldali képét, ahol a UserForm1 objektum helyi menüje látható). A törlés előtt lehetőségünk van a törölt elem exportálására (azaz önálló fájlként való kimentésére), ami egyébként a helyi menü Export File… funkciójával is megtehető.

Az Import File… funkcióval betölthetünk és az aktuális projekthez adhatunk önálló fájlokban lévő formokat (*.frm), modulokat (*.bas), és osztálymodulokat (*.cls). Az export és import funkciókkal tehát könnyen tudunk objektumokat átvinni projektek között.

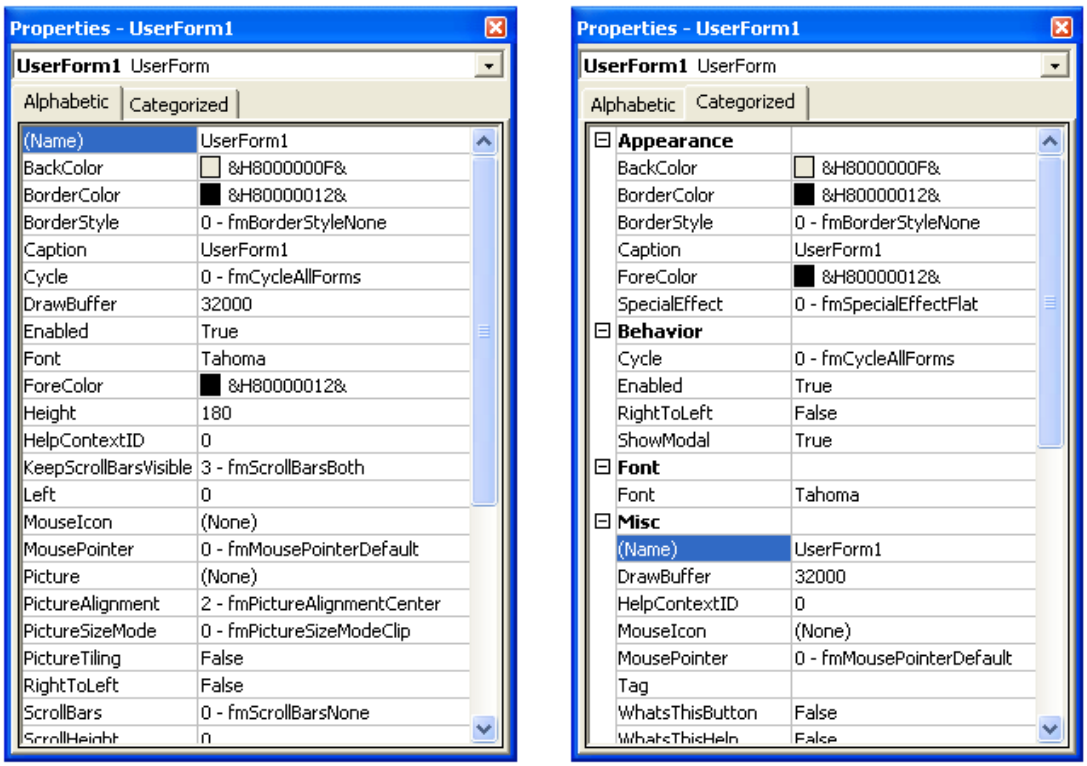
Megjegyzés

- A *.frm, *.bas, *.cls fájltípusokat nemcsak az Excel, hanem más MS Office alkalmazások, valamint az MS Visual Studio fejlesztőrendszer szoftverei is egységesen kezelik.
- A formok exportálásakor a *.frm fájlon kívül egy *.frx fájl is létrejön, amely bináris információkat tárol a formról.
- Az osztálymodulokkal nem foglalkozunk (mert ez a témakör „túlnő” a programozási alapismereteken).

3.2.3. A Properties ablak

A Properties (tulajdonságok) ablak a projekthez tartozó objektumok tulajdonságainak megjelenítésére és azok tervezéskori megadására, módosítására használatos. A vizuális formtervezés (lásd 3.2.7. fejezet) során itt adjuk meg az adott form, illetve a formon lévő vezérlők tulajdonságait (lásd 3.7., 3.11. ábra).

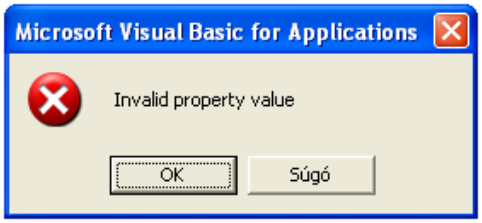
Az ablak tetején lévő legördülő listában kiválasztható az a vezérlő (a formot is beleértve), amelynek tulajdonságait látni, illetve megadni szeretnénk. A tulajdonságokat kétféle sorrendben tudjuk megjeleníteni (a megfelelő fül kiválasztásával): névsor szerint (Alphabetic fül), vagy a tulajdonságokat kategóriák szerint csoportosítva (Categorized fül). Ha már tudjuk az adott tulajdonság nevét, akkor a névsor szerinti fülön gyorsan megtaláljuk, különben célszerű a kategorizált fülön keresgélni, hiszen ott az összes tulajdonság helyett elegendő azon csoport végignézése, amelyikhez a keresett tulajdonság tartozik.



3.11. ábra. A Properties ablak

A Properties ablakban (mindkét fül esetén) két oszlop látható, a bal oldali a tulajdonságok neveit, a jobb oldali azok aktuális értékeit tartalmazza. Az adatok megadása a jobb oldali oszlopban történik, ahol is az adott tulajdonságtól függően vagy begépeljük a tulajdonság értékét (pl. Name, Caption, Left), vagy egy készletből kiválasztjuk (pl. Enabled, MousePointer, PictureAlignment), vagy egy párbeszédablakkal adjuk meg (pl. Font, Picture).

A begépeléssel megadott adatok értéke ellenőrződik, érvénytelen adat esetén (pl. szám helyett szöveget adunk meg) hibaüzenetet kapunk (lásd 3.12. ábra).



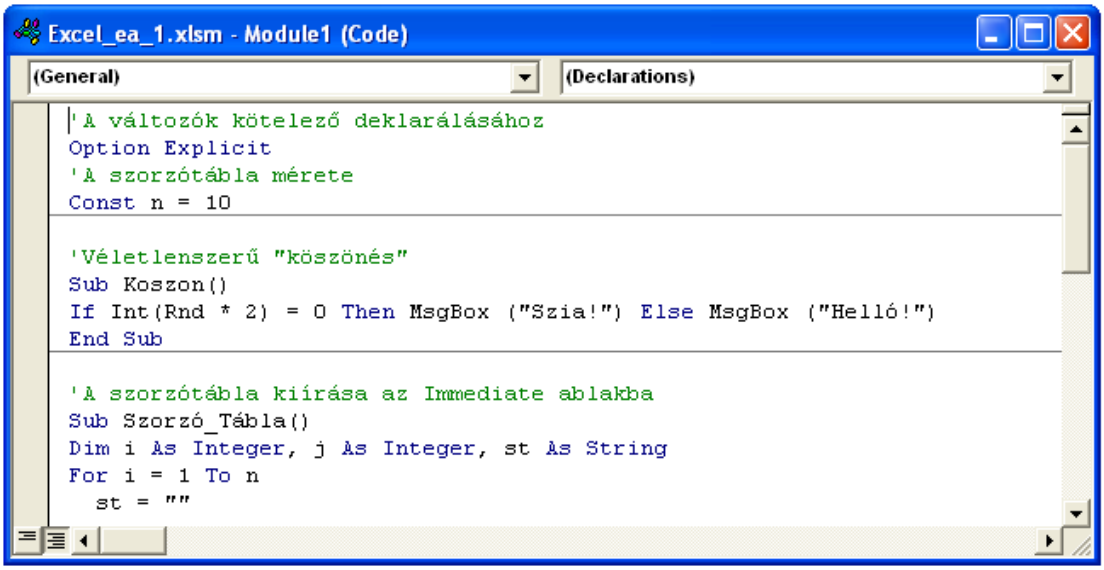
3.12. ábra. Érvénytelen tulajdonság értékről tájékoztató hibaüzenet

Megjegyzés

- Az Alphabetic fülön a név (Name) tulajdonság nem a névsor szerinti helyén jelenik meg, hanem a lista elején.
- A Categorized fülön az egyes tulajdonság csoportok (mint pl. megjelenés (Appearance), viselkedés (Behavior)) az előttük megjelenő (mínusz, plusz) ikonokkal becsukhatók (egysorossá), illetve kinyithatók.
- A tulajdonságok nevéből általában már kiderül a szerepük, de a tulajdonság ablakban is „él" a környezetfüggő súgó, így az aktuálisan kiválasztott tulajdonság súgójához elegendő leütni az F1 billentyűt.
- Bizonyos tulajdonságok (pl. BackColor, ForeColor) értéke többféle (pl. begépelés, választás) módon is megadható.
- Csoportos kijelölés esetén (lásd 3.2.7. fejezet) a Properties ablakban a kijelölt vezérlők közös tulajdonságai jelennek meg (pl. a Name tulajdonság ekkor nem látható), és ezek egyszerre módosíthatók.
- Az ablak tetején lévő legördülő listában olyan vezérlő is kiválasztható, amely nem jelenik meg a formon tervezéskor, így nem tudunk rákattintani (pl. ha véletlenül olyan pozíciót (lásd Left, Top tulajdonságok) adtunk meg, amivel a vezérlő „lekerült" a formról).
- Vannak olyan tulajdonságok, amelyek csak futási időben érhetők el, és vannak olyanok is, amelyek csak olvashatók (read-only) (pl. ListBox.ListCount).
- A fontosabb tulajdonságokról a vizuális formtervezésben (lásd 3.2.7. fejezet) lesz szó.

3.2.4. A kódszerkesztő ablak

A kódszerkesztő ablak a VBA forráskódok megírására, azok módosítására szolgál. A forráskódok modulokba, azon belül pedig szubrutinokba szervezettek. A kódszerkesztő ablak tetején két legördülő lista található, a bal oldaliban az objektumok, a jobb oldaliban a szubrutinok nevei jelennek meg, illetve választhatók ki.

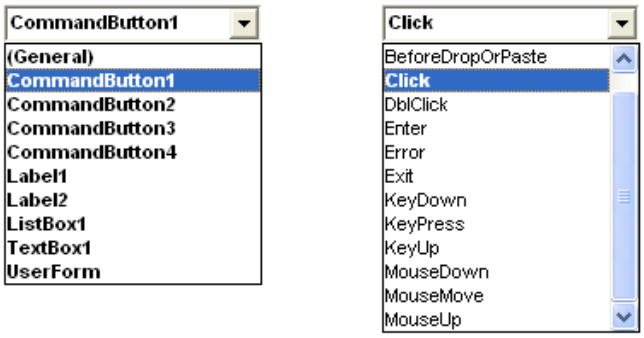


3.13. ábra. A kódszerkesztő ablak

A listák tartalma egyfelől igazodik a kurzor aktuális (forráskódbeli) helyéhez, másfelől segítségükkel gyorsan rápozícionálhatunk a modul kívánt részére. Választáskor általában először a bal oldali listából, utána a jobb oldaliból választunk, mert a jobb oldali lista tartalma a bal oldali lista aktuális eleméhez igazodik.

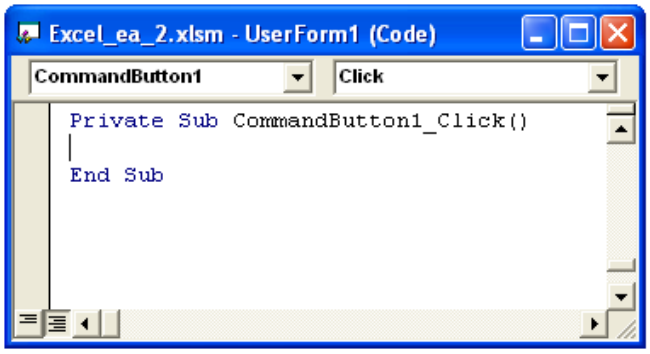
A bal oldali listában az első elem mindig egy (General) nevű elem. Ha ez az elem van kiválasztva, akkor a jobb oldali listában az első elem egy (Declarations) nevű elem (lásd 3.13. ábra), a többi elem pedig a modulban található általános (nem eseménykezelő) szubrutinok nevei. A (Declarations) elem kiválasztása a

modul elejére pozícionál (azaz a deklarációs részre, ahol a modulszintű utasítások találhatók, lásd 3.2.5. fejezet), egyébként meg a kiválasztott szubrutin elejére.



3.14. ábra. A legördülő listák tipikus tartalma egy formhoz tartozó modul esetén

A bal oldali listában (a (General) elemen kívül) az adott modul objektumai (azok nevei) jelennek meg (pl. egy form moduljában a form és a rajta található vezérlők (lásd 3.14. ábra), a munkafüzethez tartozó modulban a munkafüzet objektum). Ha egy objektum van kiválasztva a listában (és nem a (General) elem), akkor a kiválasztott objektumhoz tartozó eseménykezelők (mint szubrutinok) nevei jelennek meg a jobb oldali listában (lásd 3.14. ábra). A már definiált eseménykezelők nevei félkövéren jelennek meg. Egy még nem definiált eseménykezelő kiválasztása létrehozza az adott eseménykezelőt üres tartalommal, amit szerkeszthetünk (lásd 3.15. ábra), egyébként meg a kiválasztott eseménykezelő elejére ugrik a kurzor a kódszerkesztő ablakban.

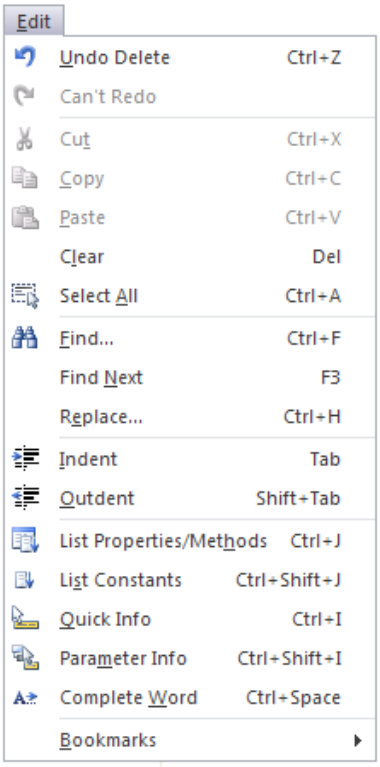


3.15. ábra. Egy létrehozott új eseménykezelő

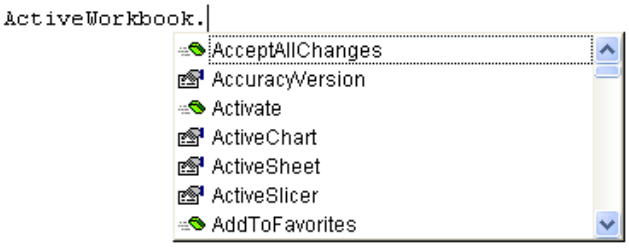
Megjegyzés

- Ha a jobb oldali legördülő listában egy eseménykezelő van kiválasztva, és a bal oldali listában egy másik objektumot választunk ki, akkor, ha van az adott objektumnak olyan eseménykezelője, akkor arra pozícionálunk, ha nincs, akkor létrejön az objektum ezen eseménykezelője. Az így létrejött (esetleg felesleges) üres eseménykezelők (amik a futást nem befolyásolják) természetesen törölhetők.
- A kódszerkesztő ablakok két részre oszthatók a Window menü Split funkciójával (lásd 3.8. ábra), így a forráskód két tetszőleges részét láthatjuk, illetve szerkeszthetjük.

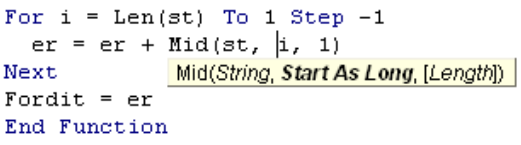
A kódszerkesztő „viselkedése” a hozzá tartozó beállításoktól (lásd 3.20. ábra) is függ. Ezektől a beállításoktól függetlenül a kódszerkesztőben lehetőség van (többek között, lásd 3.16. ábra) egy objektum tulajdonságainak és metódusainak a megjelenítésére (List Properties/Methods, lásd 3.17. ábra), a szubrutinok paraméterezésének megjelenítésére (Parameter Info, lásd 3.18. ábra), valamint a szókiegészítésre (Complete Word, lásd 3.19. ábra) is.



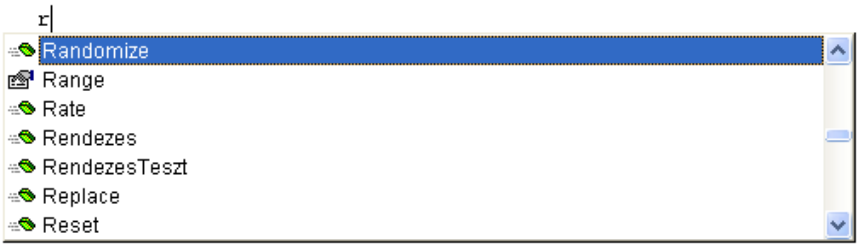
3.16. ábra. A Visual Basic Editor Edit menüje



3.17. ábra. A List Properties/Methods funkció

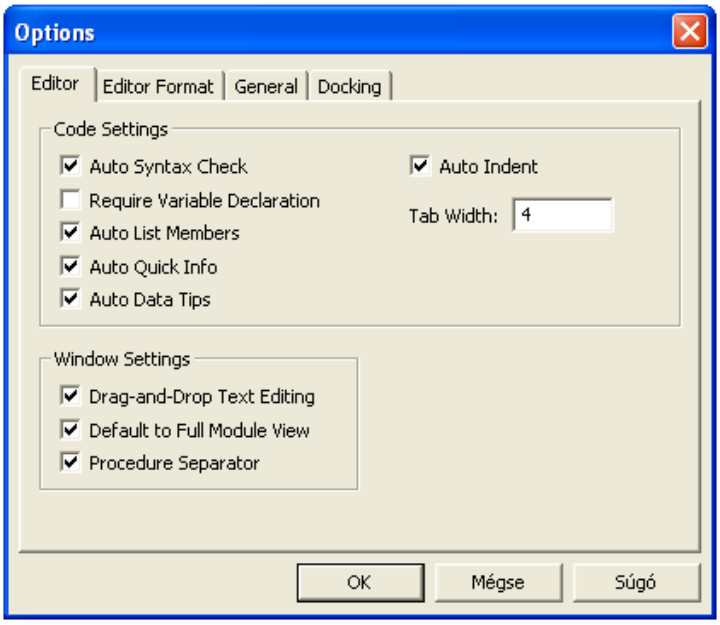


3.18. ábra. A Parameter Info funkció



3.19. ábra. A Complete Word funkció

A Visual Basic Editor „testreszabása” a Tools menüben található Options… funkcióval tehető meg. A funkció egy párbeszédablakot jelenít meg (lásd 3.20. ábra), ahol egyrészt megtekinthetők, másrészt módosíthatók az egyes beállítások. Az első két fül (Editor, Editor Format) a kódszerkesztő beállításait tartalmazza, a General fülön általános beállításokat végezhetünk, míg a Docking fülön az ablakok rögzíthetőségét szabályozhatjuk.



3.20. ábra. A Visual Basic Editor beállításainak párbeszédablaka

Az ablak négy fülén beállítható értékekből csak néhány, a kódszerkesztő működésére vonatkozó beállítást ragadunk ki. Az Auto Syntax Check kapcsoló a kódszerkesztő ablakban megadott forráskód sorainak automatikus szintaktikai ellenőrzését szabályozza. Bekapcsolt állapotban egy szintaktikailag helytelen sor elhagyásakor hibaüzenetet kapunk, és a sor a megfelelő színnel (az Editor Format fülön megadott Syntax Error Text alapján, ami alapértelmezésben piros betűszínű) kiemelődik.

Az Auto List Members kapcsoló az automatikus kódkiegészítést szabályozza. Bekapcsolt állapotban a forráskód begépelésekor megjelenik egy olyan lista a képernyőn, amely logikailag kiegészítheti az adott utasítást (pl. egy objektum után leütött pont hatására kiajánlódnak az objektum megfelelő tulajdonságai, metódusai).

Az Auto Quick Info kapcsoló bekapcsolt állapotában a szubrutinok paraméterezéséről kapunk egy felbukkanó súgót kódszerkesztés közben, ahol az éppen megadandó paraméter félkövéren jelenik meg.

Megjegyzés

- Célszerű megtartani az alapértelmezett beállításokat (amelyben pl. a fent említett kapcsolók mindegyike bekapcsolt állapotú).
- A forráskód szerkesztésekor a szintaktikailag helyes sor néha „átalakul”, amint a kurzor elhagyja az adott sort. A kulcsszavak kiemelődnek (alapértelmezésben kék színnel), a felismert (azaz nem elgépelt) azonosítók a deklaráláskor megadott alakban (kis- és nagybetű) jelennek meg, esetlegesen szóközök szúródnak be, illetve felesleges szóközök törlődnek. Ez az „egységesített küllem” javítja az áttekinthetőséget, a forráskód olvashatóságát.
- Az objektumok tulajdonságait és metódusait célszerű a kódkiegészítéssel kiajánlott listából kiválasztani, mert gyorsabb, és így biztosan nem gépeljük el.
- Az egyes szubrutinokat (valamint a deklarációs részt) alapértelmezésben egy vízszintes vonal választja el egymástól (aminek megjelenését a Procedure Separator kapcsoló szabályozza (lásd 3.20. ábra)).

3.2.5. Modulok felépítése

Minden modul, legyen az objektumhoz tartozó vagy önálló modul (lásd 3.2.2. fejezet), ugyanazt az egyszerű felépítést követi. A modul elején a modulszintű utasításokat kell megadni, amelyeket a modul szubrutinjai követnek.

Modulszintű utasítások

- Deftype utasítások (pl. `DefInt`)
- Opciókat megadó utasítások (pl. `Option Explicit`)
- Típusdeklarációk (pl. `Type`, `Enum`)
- Modulszintű konstansok (`Const`), illetve változók (`Public`, `Private`, `Dim`) deklarálása

Megjegyzés: Az opciókat megadó utasításokkal a Visual Basic fordító/értelmező működését szabályozhatjuk. Ezek, illetve a Deftype utasítások érvényessége (scope) csak az adott modulra terjed ki. A modulszintű típus, konstans és változó deklaráció azonban deklarálhat más modulból is hivatkozható (publikus) elemeket is.

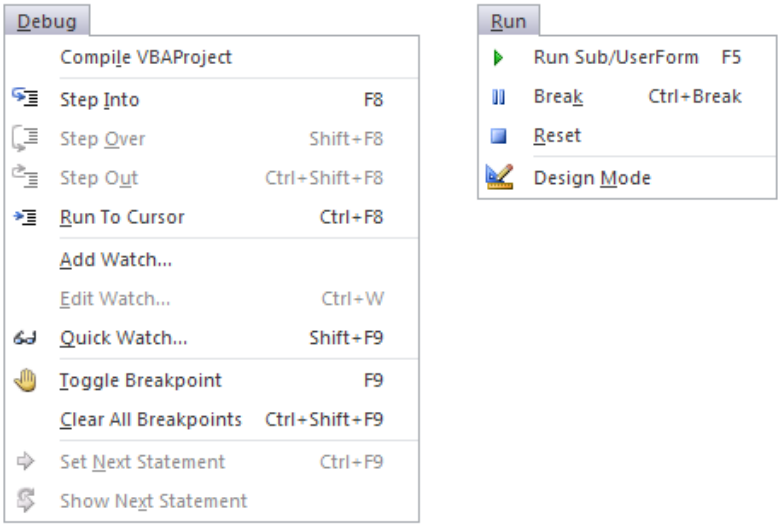
Pl.

```
'A változók kötelező deklarálásához
Option Explicit

'A szorzótábla mérete
Const n = 10

'Véletlenszerű "köszönés"
Sub Koszon()
If Int(Rnd * 2) = 0 Then MsgBox "Szia!" Else MsgBox "Helló!"
End Sub

'A szorzótábla kiírása az Immediate ablakba
Sub Szorzo_Tabla1()
Dim i As Integer, j As Integer, st As String
For i = 1 To n
  st = ""
  For j = 1 To n
    st = st + " " & i & "*" & j & "=" & i * j
  Next
  Debug.Print st
Next
End Sub

'A szorzótábla egy másfajta kiírása az Immediate ablakba
Sub Szorzo_Tabla2()
'Egy adat mezőszélessége
Const msz = 5
Dim i As Integer, j As Integer
For i = 1 To n
  Debug.Print Tab(i * msz + 1); i;
Next
Debug.Print
For i = 1 To n
  Debug.Print i;
```

```
  For j = 1 To n
    Debug.Print Tab(j * msz + 1); i * j;
  Next
  Debug.Print
Next
End Sub
```

3.2.6. Fordítás, futtatás, hibakeresés

A forrásprogramok írásakor a kódszerkesztő már ellenőrizni tudja az adott sor szintaktikáját (lásd 3.2.4. fejezet). A Visual Basic Editor Debug menüjének Compile VBAProject funkciójával (lásd 3.21. ábra) az adott projekthez tartozó összes forrásprogram lefordítható. Ezzel kiszűrhetők a forrásprogramok szintaktikai (formai) hibái, ugyanis az első megtalált hibáról üzenetet kapunk, a kódszerkesztő ablakban ez a kódrészlet jelenik meg, és a hibás sor kiemelődik.

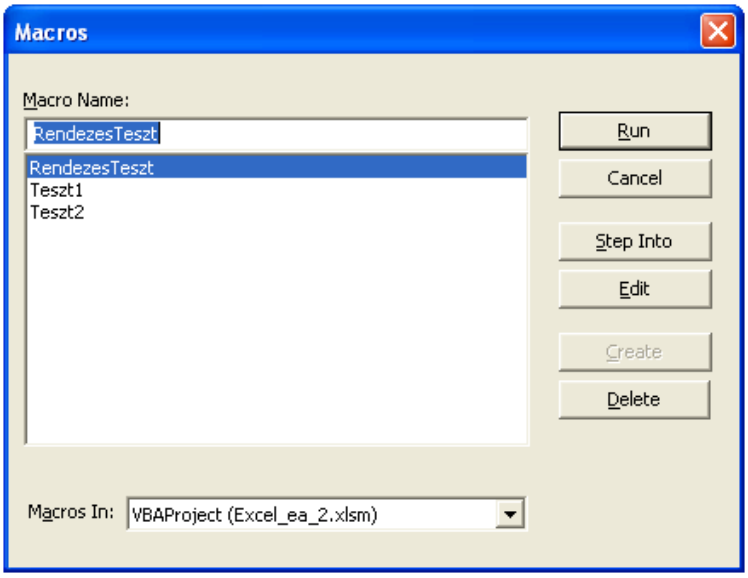


3.21. ábra. A Visual Basic Editor Debug és Run menüje

A forráskódok természetesen nemcsak fordíthatók, de futtathatók is. A futtatás funkció a Run menü Run Sub/UserForm funkciójával (lásd 3.21. ábra) végezhető. A funkció neve is utal arra, hogy egy szubrutint, vagy egy felhasználó által készített formot (UserForm) lehet futtatni, és a kettő között különbség van, ugyanis, ha egy formhoz tartozó modulban kérjük ezt a funkciót (pl. az F5 gyorsbillentyűvel), akkor függetlenül attól, hogy a modul mely részén áll a kurzor, a form futtatását kérjük. Az ilyen modulban lévő szubrutinok tehát nem futtathatók külön-külön. A futtatás funkció nem fordítja le az eseménykezelő szubrutinokat.

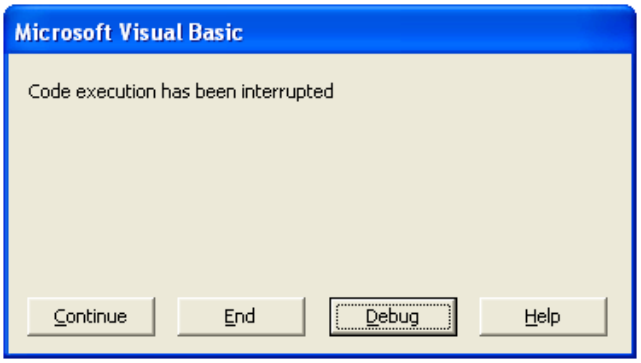
Más a helyzet a többi (nem formhoz tartozó) modullal. Itt ugyanis minden olyan szubrutin külön futtatható, amelynek nincsen paramétere. Álljunk a kurzorral a futtatni kívánt szubrutin egy sorára és kérjük a futtatás funkciót. Ekkor a szubrutin fordítása is megtörténik. A paraméterrel rendelkező szubrutinokat nem lehet közvetlenül futtatni, ezek csak közvetve, valamilyen őket meghívó szubrutin segítségével futtathatók.

Ha az aktuális sor nem tartozik egyik szubrutinhoz sem (pl. két szubrutin közötti üres soron vagy a modul deklarációs részében áll a kurzor), akkor a futtatás funkció egy párbeszédablakot jelenít meg (ugyanezt teszi a Tools menü Macros… funkciója is), amelyben a modulban található futtatható (paraméter nélküli) szubrutinok listája jelenik meg (lásd 3.22. ábra). Az ablakban kiválasztható a futtatandó (Run), lépésenként futtatandó (Step Into), szerkesztendő (Edit), vagy törlendő (Delete) szubrutin, amely akár egy másik projektben is lehet (mivel a Macros In legördülő lista segítségével akár az összes nyitott projekt futtatható szubrutinjának nevét is megjeleníthetjük a makrókat tartalmazó listában).



3.22. ábra. A futtatható szubrutinok párbeszédablaka

Egy program futásának megszakítása a Run menü Break funkciójával kérhető, amelynek gyorsbillentyűje Ctrl+Break vagy Esc (pl. egy végtelen ciklusba esett program futását csak ezekkel a billentyűkkel lehet megszakítani). A megszakított programfutás esetén a futás folytatható (Continue), leállítható (End), illetve felfüggeszthető (Debug) (lásd 3.23. ábra).

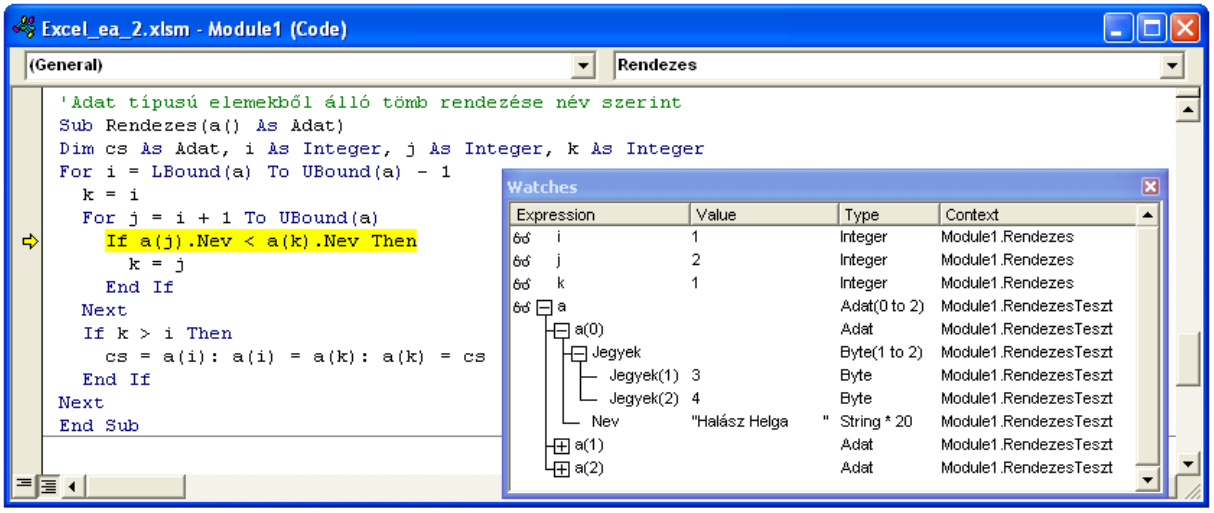


3.23. ábra. A megszakított programfutás párbeszédablaka

Egy futó (vagy futásában felfüggesztett) program futásának a befejezése a Run menü Reset funkciójával kérhető. A Run menü funkciói az eszköztársor segítségével is aktivizálhatók.

Egy program helyes működésének tesztelése a programfejlesztői munka része. Sajnos gyakran előfordul, hogy a programunk nem úgy működik, ahogy szeretnénk. A programok szemantikai (tartalmi, jelentésbeli, logikai) hibáinak feltárása már nehezebb feladat (szemben azzal, hogy a szintaktikai hibákat egyetlen fordítással kiszűrhetjük).

A programfejlesztő rendszerek (így a VBA is) speciális funkciókat biztosítanak a programok szemantikai hibáinak felderítésére. Ezek a funkciók alapvetően két fontos szolgáltatást nyújtanak: megtudhatjuk, hogy merre halad a vezérlés (milyen utasítások kerülnek végrehajtásra), és hogy az egyes változókban milyen értékek vannak.



3.24. ábra. Lépésenkénti futtatás és a Watches ablak

A programok lépésenkénti futtatása a Debug menü Step Into funkciójával végezhető. Ez azt jelenti, hogy a végrehajtás utasításonként (de mivel általában egy sorba egy utasítást írunk, ezért gyakorlatilag soronként) történik, és minden egyes utasítás végrehajtása előtt a program futása felfüggesztődik. Ekkor lehetőségünk van a változók tartalmának megtekintésére (esetleges módosítására), majd a futás folytatására, illetve a futás leállítására (ha pl. már rájöttünk az esetleges hibára). Az éppen végrehajtásra kerülő utasítás (alapértelmezésben) sárga háttérszínnel (és a margó sávon egy sárga nyíllal) emelődik ki (lásd 3.24. ábra).

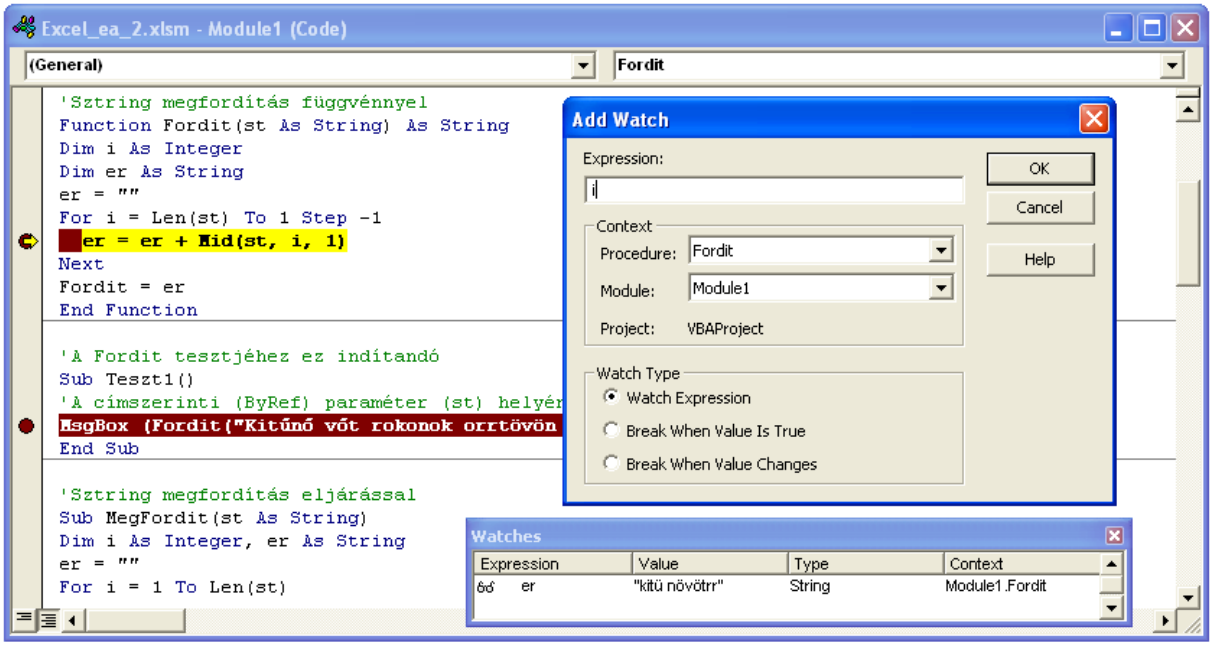
Egy hosszabb forrásprogram esetén hasznos lehet, ha csak ott kezdjük el a lépésenkénti futtatást, ahol a hibát sejtjük. Ez megtehető a forráskódban elhelyezhető töréspontok (Breakpoints) segítségével. A töréspontok adott sorra való elhelyezése, illetve levétele a Debug menü Toggle Breakpoint funkciójával végezhető. A programfutás egy töréspontra érve felfüggesztődik (azaz lehetőségünk van a lépésenkénti futtatásra, stb.). A töréspontok sorát a margón egy kör jelzi, míg a sor (alapértelmezésben) bordó háttérszínnel emelődik ki (lásd 3.25. ábra).

A forráskód egy adott soráig való futtatás (Debug menü Run To Cursor funkció) is azt a célt szolgálja, hogy ott függesszük fel a program futását, ahol a hibát sejtjük.

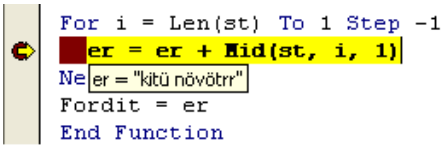
A Debug menü Step Over funkcióját akkor használjuk, ha lépésenkénti futtatáskor egy olyan utasításra lépünk, amelyik szubrutinhívást tartalmaz (pl. egy saját eljárást vagy függvényt hívunk meg), és szeretnénk ezt az utasítást egy lépésben végrehajtani (mert pl. tudjuk, hogy a hiba nem abban a szubrutinban van). A Step Into funkció ugyanis ekkor belép az adott szubrutinba, és ott folytatódik a lépésenkénti futtatás. Ha beléptünk egy szubrutinba, akkor használható a Step Out funkció is, ami lefuttatja az adott szubrutint, majd a lépésenkénti futtatás a hívó szubrutinban a hívást követő utasítással folytatódik.

A változók értékeinek megfigyelési (Watches) ablakát (lásd 3.24. ábra) a View menü Watch Window funkciójával jeleníthetjük meg. Az ablak helyi menüjével új változókat vehetünk fel (Add Watch…) (lásd 3.25. ábra), vagy egy meglévőt módosíthatunk (Edit Watch…), illetve törölhetünk (Delete Watch…).

A tömb- és rekordváltozók az előttük megjelenő (mínusz, plusz) ikonokkal becsukhatók, illetve kinyithatók így az egyes tömbelemek, illetve rekordmezők is megtekinthetők (lásd 3.24. ábra).



3.25. ábra. Egy töréspontonon felfüggesztett programfutás a Watches és az Add Watch ablakokkal



3.26. ábra. Egy változó értékének megjelenítése az egérkurzor segítségével

Megjegyzés

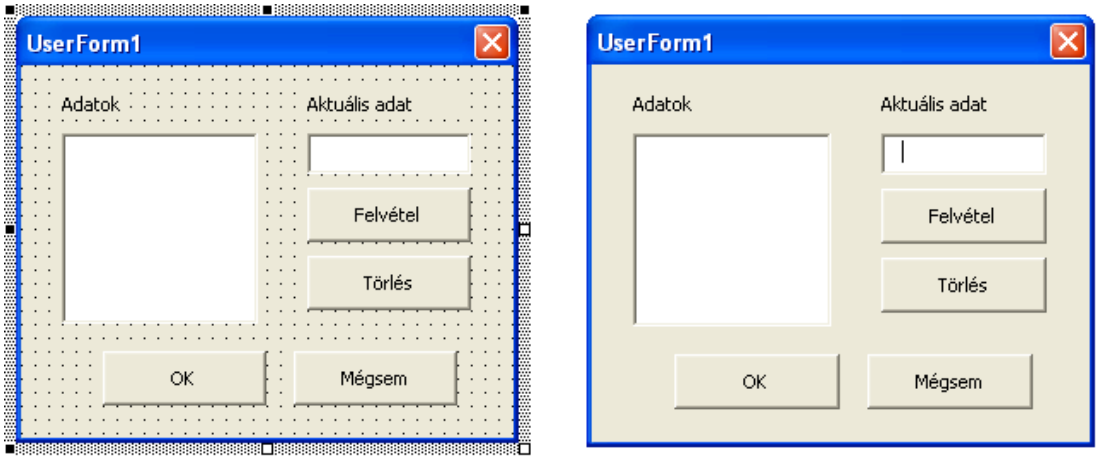
- Felfüggesztett programfutás esetén az egyszerű (nem összetett adattípusú) változók tartalma úgy is megtekinthető, hogy az egérrel az adott változó fölé pozícionálunk (lásd 3.26. ábra). Ehhez az Auto Data Tips kapcsolónak (lásd Tools menü Options… funkció, Editor fül) bekapcsolt állapotban kell lennie (lásd 3.20. ábra).
- A Watches ablakban nemcsak változók értékeit figyelhetjük meg, de kifejezéseket is kiértékelhetünk. A változók tartalmát az értékük (Value) módosításával meg is változtathatjuk.
- Töréspontokat nem helyezhetünk el akárhol (pl. üres vagy egy `Dim` utasítást tartalmazó soron nem lehet töréspont). A töréspontokat a kódszerkesztő ablak bal szélén (a margó sávon) való egérkattintással is elhelyezhetünk (illetve levehetünk).

3.2.7. Vizuális formtervezés

A Windows alkalmazásokban a felhasználóval történő kommunikáció egyik alapvető eszköze az ablak (más néven form, űrlap). Természetesen az alkalmazás helyes működése a legfontosabb, de a küllem, az áttekinthetőség, a könnyű kezelés, vagyis a felhasználóbarát viselkedés is lényeges szempont. Éppen ezért a Windows platformú szoftverfejlesztő rendszerek (így az MS Excel VBA is) támogatják a formok vizuális, interaktív tervezését.

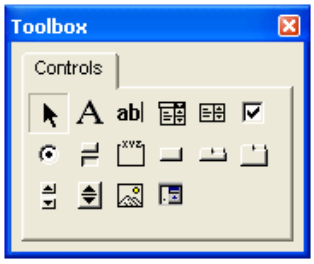
A tervezés (manapság általánosan elterjedt) jellemzője a WYSIWYG (What You See Is What You Get) tervezési mód, ami azt jelenti, hogy a tervezett ablak pontosan úgy fog megjelenni futáskor, mint ahogyan azt a tervezéskor látjuk (lásd 3.27. ábra).

Egy új form létrehozása a Visual Basic Editor Insert menüjének UserForm funkciójával (vagy a Project ablak helyi menüjének segítségével) hozható létre. A form (pl. a jobb szélén és alján lévő fehér keretező négyzetek segítségével) tetszőlegesen átméretezhető.



3.27. ábra. Egy form tervezéskor és futáskor

A formra tehető, használható vezérlőket (Controls) a Toolbox ablak jeleníti meg (lásd 3.28. ábra). Egy form tervezésekor (azaz ha a UserForm objektum ablaka aktív), a Toolbox ablak automatikusan megjelenik (és ha az ablakot esetleg bezárnánk, akkor a View menü Toolbox funkciójával, vagy az eszköztársor megfelelő ikonjával újra megnyithatjuk).



3.28. ábra. A Toolbox ablak

A Toolbox ablak a fontosabb Windows vezérlőket tartalmazza, mint pl. címke (Label), beviteli mező (TextBox), legördülő lista (ComboBox), lista (ListBox), jelölőnégyzet (CheckBox), választógomb (OptionButton), nyomógomb (CommandButton).

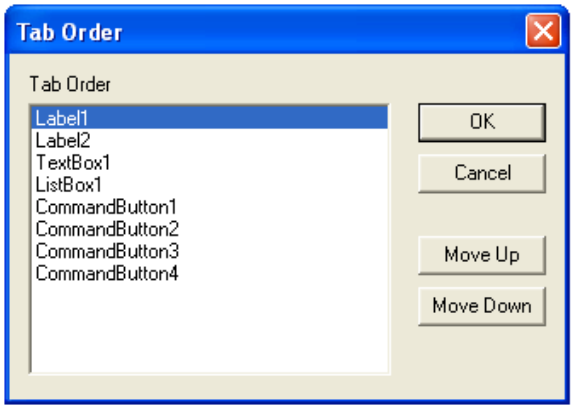
Egy vezérlő formra helyezése az egérrel történhet. A kívánt vezérlő a Toolbox ablakból a formra vihető a „fogd és vidd" egérművelettel, de a kívánt vezérlőn, majd a formon való kattintás is ugyanezt teszi, azaz elhelyezi a formon az adott vezérlő egy példányát (alapértelmezett méretben). Ha a formon kattintás helyett egy téglalap alakú területet jelölünk ki, akkor a vezérlő a megadott méretben kerül a formra.

A form, illetve a formon lévő vezérlők egérkattintással kijelölhetők, de a Windows-ban szokásos kijelölési technikákkal (lenyomott Ctrl vagy a Shift billentyű és egérkattintás, vagy jelölőkeret rajzolásával) csoportos elemkijelölés is végezhető. Csoportos kijelölést akkor használunk, ha a vezérlők egy csoportjára szeretnénk valamilyen műveletet (pl. mozgatás, törlés, adatmegadás) elvégezni.

A kijelölt vezérlők az egér „fogd és vidd" művelettel szabadon mozgathatók (áthelyezhetők), és méretezhetők (ehhez valamelyik keretező négyzetet kell mozgatnunk). Ha több vezérlő van kijelölve, akkor a Properties ablakban csak a közös tulajdonságok jelennek meg.

A kijelölt vezérlő (vagy vezérlők) kezelése az Edit menü (vagy a helyi menü) segítségével is történhet. A törlés a Delete billentyűvel (illetve a Delete funkcióval) végezhető, de a vágóasztal többi művelete (pl. vágólapra másolás, beillesztés) is használható.

Futáskor alapesetben az egyes vezérlők a formra való felkerülésük sorrendjében aktivizálhatók. Az első vezérlő lesz fókuszban, és a felhasználó a Tab (illetve Shift+Tab) billentyűkkel tud ezen sorrend szerint „lépkedni" az egyes vezérlőkön. Ez azonban nem feltétlenül igazodik a vezérlők formon való elrendezéséhez. A form helyi menüjének Tab Order funkciójával módosítható a vezérlők futáskori sorrendje (lásd 3.29. ábra). Az aktuálisan kiválasztott vezérlő felfelé, illetve lefelé mozgatható a megfelelő nyomógombokkal (Move Up, Move Down), így a megfelelő sorrend kialakítható.



3.29. ábra. A Tab Order ablak

A formra tett vezérlők a vezérlő típusából és egy sorszámból álló azonosítót (Name) kapnak (pl. CommandButton1). A sorszám egytől kezdődően kerül kiosztásra. A kifejezőbb, olvashatóbb programkód érdekében a vezérlőket át is nevezhetjük, ezt célszerű közvetlenül a formra való elhelyezés után megtenni, még mielőtt eseménykezelőket definiálnánk hozzá (ugyanis a megváltozott neveket az eseménykezelők nevei nem követik).

Egy vezérlő egy eseménykezelőjének létrehozása (illetve már meglévő eseménykezelő esetén az arra való rápozícionálás) általánosan a kódszerkesztő ablak tetején lévő legördülő listákból való választással történhet. Azonban a vezérlők alapértelmezett eseménykezelője (ami pl. nyomógomb esetén a kattintásnak megfelelő Click, beviteli mező esetén a tartalom megváltozásának megfelelő Change) a vezérlőn való dupla kattintással is megnyitható.

Az eseménykezelők azonosítója az adott objektum azonosítójából (Name), az alulvonás karakterből, és az esemény nevéből áll, a paraméterezésük pedig rögzített (pl. `CommandButton1_Click()`, `ListBox2_DblClick(ByVal Cancel As MSForms.ReturnBoolean)`).

Megjegyzés

- Egy vezérlő a Properties ablak legördülő listájával is kijelölhető.
- Az objektumoknak lehetnek olyan tulajdonságai is, amelyek csak futási időben érhetők el, ezek meg sem jelennek a Properties ablakban (pl. ListBox.ListIndex).
- Néhány fontosabb tulajdonság: név (Name), felirat (Caption), elhelyezkedés (Top, Left), méret (Width, Height), választhatóság (Enabled), láthatóság (Visible).

3.2.8. Mintafeladat

Ebben a fejezetben egy nem túl bonyolult, de azért már kellően összetett feladat megoldását mutatjuk be. Az egyszerűség és a könnyebb követhetőség érdekében a teljes feladatot több részfeladatra bontottuk.

1. Feladat: Készítsünk adatbeviteli formot több adat megadására!

Megoldás: Ahhoz, hogy több adat megadható legyen, biztosítanunk kell egy olyan vezérlőt, amelyben egy adat megadható, és egy olyat, amelyben az eddig megadott adatok megtekinthetők. Egy beviteli mezőt és egy lista vezérlőt fogunk használni. A beviteli mezőben megadott adat felvételét, elfogadását egy (Felvétel feliratú) nyomógombbal kérhetjük. A két vezérlőt címkékkel is ellátjuk (lásd 3.27. ábra).

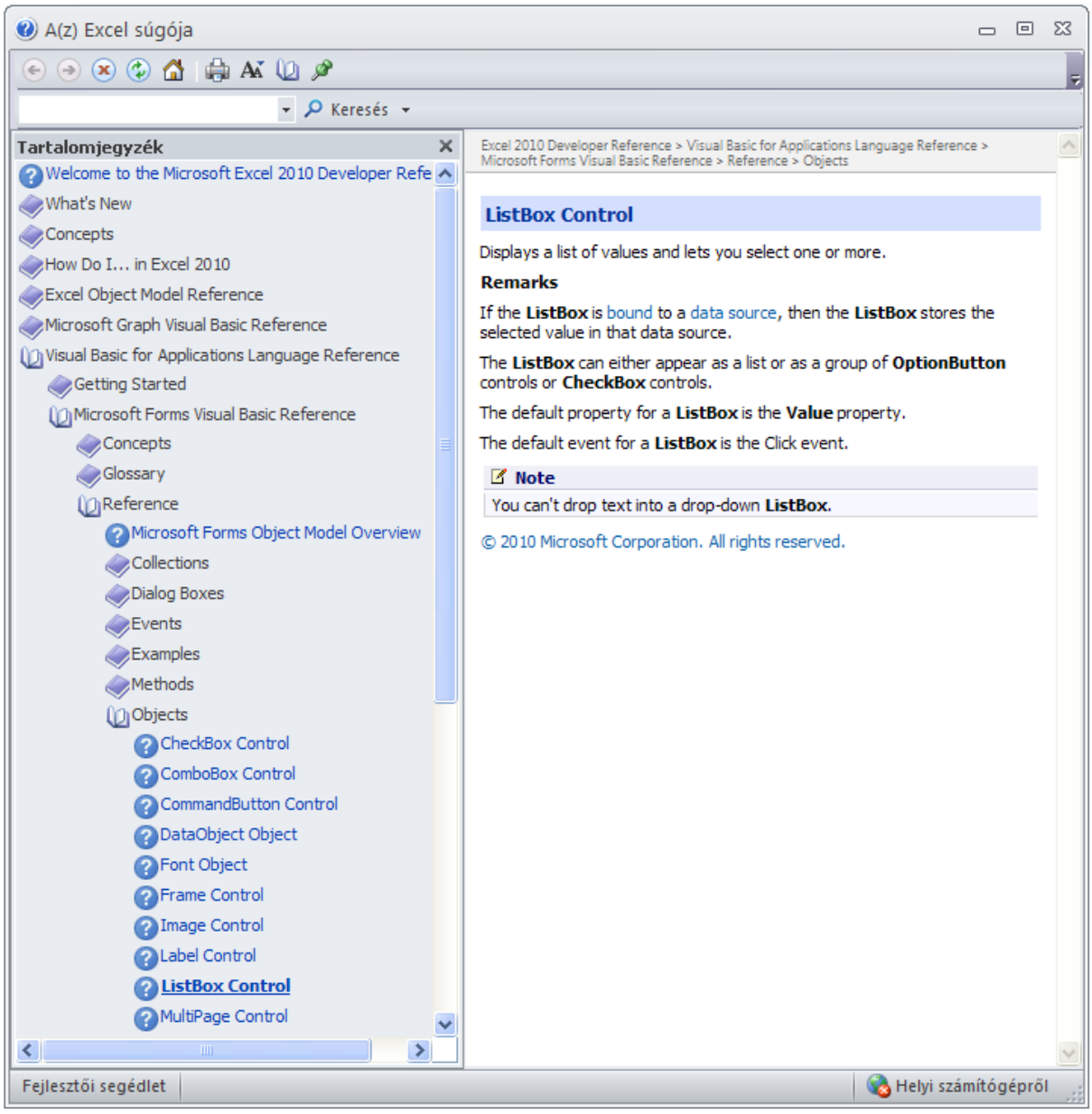
Elvégzendő tevékenységek:

- A form létrehozása (Insert menü, UserForm funkció).
- A vezérlők formon való elhelyezése (kattintás a megfelelő vezérlőn, majd a formon, esetleges mozgatás, méretezés).
- Feliratok megadása (Caption tulajdonságok a Properties ablakban).
- A Felvétel gomb kattintás eseménykezelőjének megírása (pl. duplakattintás a Felvétel nyomógombon létrehozza az üres eseménykezelőt). Az eseménykezelőben a lista elemeit kell bővíteni a beviteli mezőben megadott adattal.

Az eseménykezelő forráskódja:

```
'Felvétel (minden adatot felvesz)
Private Sub CommandButton1_Click()
ListBox1.AddItem TextBox1.Text
End Sub
```

A feladat megoldásához szükséges információkhoz (nevezetesen, hogy hogyan hivatkozzuk a beviteli mezőben megadott adatot, és hogyan lehet egy lista elemeit bővíteni) a súgó, illetve az objektumtallózó (Object Browser) segítségével juthatunk.

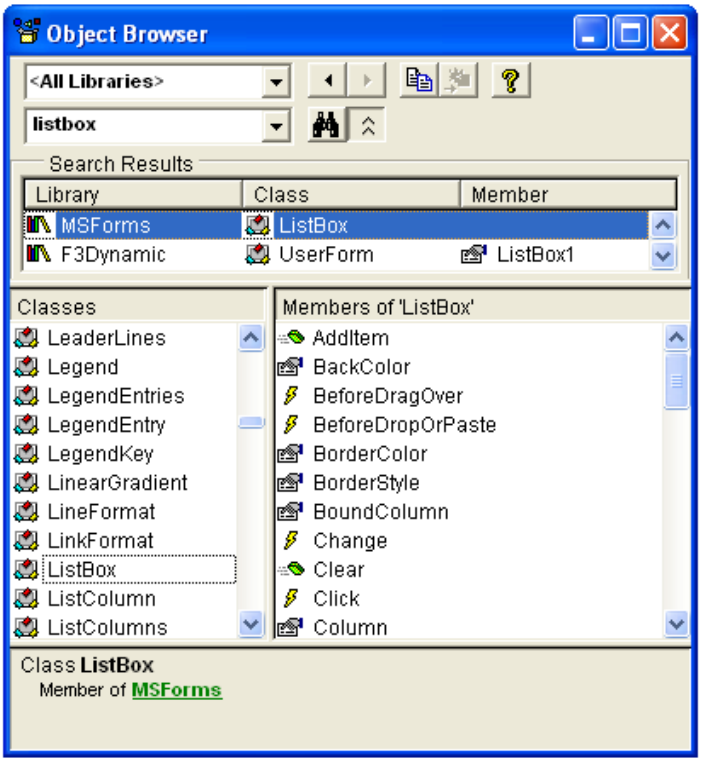


3.30. ábra. A Visual Basic Editor súgója

Ha pl. tervezéskor kijelöljük a lista vezérlőt és lenyomjuk az F1 billentyűt, akkor a 3.30. ábrán látható súgótartalom jelenik meg. A súgó a Windows-ban megszokott módon használható (pl. kereshetünk benne, a megnézett oldalakra visszamehetünk, stb.)

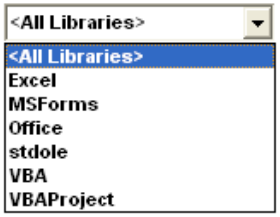
Az objektumtallózó a View menü Object Browser funkciójával jeleníthető meg. Itt (lásd 3.31. ábra) egyben láthatjuk az osztályok (így az egyes vezérlők) tulajdonságait (pl. BackColor), metódusait (pl. AddItem), és eseménykezelőit (pl. BeforeDragOver). Az ablakban keresési és súgó funkció is használható. A súgó itt is

környezetfüggő (azaz a kérdőjel ikonnal (vagy az F1 leütésével) éppen a 3.30. ábrán látható súgótartalom jeleníthető meg, de ha a jobb oldali listában választunk ki egy elemet (pl. AddItem), akkor az ahhoz tartozó súgóoldal jelenik meg).



3.31. ábra. Az Object Browser

Az Object Browser ablak (bal felső sarkában található) Project/Library legördülő listájával kiválasztható az a projekt, illetve rendszerkönyvtár (library), amelynek tartalmát megjeleníteni szeretnénk (lásd 3.32. ábra). A 3.31. ábrán az All Libraries elem van kiválasztva, így minden elérhető osztály megjelenik a Classes listában.



3.32. ábra. Az Object Browser ablak Project/Library legördülő listájának tartalma

2. Feladat: Ne fogadjunk el üres adatot!

Megoldás: A Felvétel gomb kattintás eseménykezelőjének módosítása. Üres adatnak vesszük azokat az adatokat is, amelyekben csak szóközök szerepelnek. Az adat kezdő és záró szóközeinek eltávolítására a Trim függvényt használtuk.

Az eseménykezelő forráskódja:

```
'Felvétel (üres adatot nem vesz fel)
Private Sub CommandButton1_Click()
Dim st As String
st = Trim(TextBox1.Text)
If st = "" Then
  MsgBox "Üres adat!"
```

```
Else
  ListBox1.AddItem st
End If
End Sub
```

3. Feladat: Egy korábban megadott adatot lehessen törölni!

Megoldás: Egy új nyomógomb (Törlés) formra helyezése, felirat megadása, és a gomb kattintás eseménykezelőjének megírása. Egy listában kiválasztott elem indexét a ListIndex tulajdonság mutatja (−1 esetén nincs a listában kiválasztott elem). Az első elem indexe 0, a másodiké 1, …, az utolsóé ListCount−1 (ahol a ListCount tulajdonság a lista elemeinek számát adja). A törlést a lista RemoveItem metódusával végezzük.

Megjegyzés: A megoldásban kihasználtuk, hogy a listánkban (alapértelmezésben) egyszerre legfeljebb egy elem lehet kiválasztva. A többszörös elemkiválasztást megengedő listák esetén a Selected tulajdonságot kell használni az elemek kiválasztottságának lekérdezéséhez. Azt, hogy egy lista egyszeres vagy többszörös elemkijelölést enged-e meg, a MultiSelect tulajdonság szabályozza.

Az eseménykezelő forráskódja:

```
'A listában kiválasztott adat törlése
Private Sub CommandButton2_Click()
If ListBox1.ListIndex = -1 Then
  MsgBox "Nincs kiválasztott adat!"
Else
  ListBox1.RemoveItem ListBox1.ListIndex
End If
End Sub
```

4. Feladat: Legyen a formon egy OK és egy Mégsem feliratú nyomógomb! Az OK megnyomására írjuk ki a megadott elemek számát, a Mégsem megnyomására zárjuk be a formot!

Megoldás: Két új nyomógomb (OK, Mégsem) formra helyezése, feliratok megadása. A listában lévő elemek száma a lista ListCount tulajdonságával kapható meg. A form bezárására az `End` utasítást használjuk, ami befejezi a program futását.

Az eseménykezelők forráskódja:

```
'OK gomb
Private Sub CommandButton3_Click()
MsgBox "A megadott adatok száma:" & ListBox1.ListCount
End Sub

'Mégsem gomb
Private Sub CommandButton4_Click()
End
End Sub
```

5. Feladat: Ne lehessen két egyforma adatot megadni!

Megoldás: Egy keresőciklus segítségével megvizsgáljuk, hogy szerepel-e már a felvett adatok (azaz a lista elemei) között a beviteli mezőben megadott (kezdő és záró szóközöket már nem tartalmazó) adat, ha igen, akkor üzenetet adunk, egyébként meg felvesszük a lista elemei közé.

A keresésre (egy elöltesztelő) `While` ciklust használunk. A ciklus sorban megvizsgálja a lista elemeit az első elemtől kezdve, legrosszabb esetben egészen az utolsóig. Ha megtaláljuk a keresett adatot, akkor kilépünk a ciklusból.

```
'Felvétel (nem vesz fel üres vagy már létező adatot)
Private Sub CommandButton1_Click()
Dim st As String, i As Integer, van As Boolean
st = Trim(TextBox1.Text)
If st = "" Then
  MsgBox "Üres adat!"
Else
  'Szerepel-e már a kijelölt elem a ListBox1-ben?
  van = False: i = 1
  While Not van And (i <= ListBox1.ListCount)
    If st = ListBox1.List(i - 1) Then van = True Else i = i + 1
  Wend
  If van Then
    MsgBox "Van már ilyen adat!"
  Else
    'Nem szerepel még, felvesszük
    ListBox1.AddItem st
  End If
End If
End Sub
```

Megjegyzés

- A keresőciklusban az `i` változó értékét 1-től indítottuk ListCount-ig, és a lista elemeire való hivatkozásnál korrigáltunk (hogy jó legyen az elemekre történő hivatkozás). Természetesen 0-tól ListCount−1-ig is lépkedhettünk volna, ekkor az `i`-edik (és nem az `i-1`-edik) elemre kellene a ciklusban hivatkozni.
- A keresésre azért használtunk elöltesztelős ciklust, mert a lista lehet üres is. Ekkor ugyanis a ciklusmagban a listaelemre történő hivatkozás (futási) hibát eredményezne, azaz a ciklusmagot üres lista esetén nem szabad végrehajtani.
- Az alkalmazott kereső algoritmust soros (lineáris, szekvenciális) keresésnek nevezik, mert sorban vizsgálja meg azokat az adatokat, amelyek között keresünk. Ha esetleg rendezett adatok között kellene keresni, akkor egy hatékonyabb, ún. bináris keresés (lásd szakirodalom, pl. [3]) is használható lenne. Ez a keresés ugyanis egy sikertelen hasonlítás után felére csökkenti azon elemek számát, amelyek még szóbajöhetnek az elem megtalálását illetően. A soros keresés csupán egyetlen (az éppen megvizsgált és nem egyező) elemmel csökkenti a találatra esélyes elemek halmazát.

4. AZ EXCEL PROGRAMOZÁSA

Az eddigi fejezetek általános ismereteket adtak a Visual Basic nyelvről, a Visual Basic Editor fejlesztőkörnyezetről, a vizuális tervezésről, és általában a fejlesztés mikéntjéről, de nem használták az Excel objektumait, a VBA-ban „készen kapott" objektumhierarchia lehetőségeit. Ebben a fejezetben a fontosabb Excel objektumokról, és azok forráskódból történő használatáról lesz szó.

4.1. A használható objektumok, objektumtípusok

Az objektumokhoz kötődő alapfogalmakról már olvashattunk (lásd 2.6. fejezet), most az Excel VBA környezetben használható objektumokról és objektumtípusokról lesz szó.

A fejlesztőkörnyezetben egységesen használható osztályokat (illetve modulokat, szubrutinokat, típusokat) az őket tároló fájlok (illetve rendszerfájlok) csoportosítják (lásd 3.32. ábra):

- Excel osztályok (pl. Workbooks, Range, Chart, Dialogs)
- MS Forms osztályok (pl. TextBox, ListBox, UserForm)
- MS Office osztályok (pl. CommandBar, CommandBarControl)
- VBA osztályok (pl. Collection)
- VBA Project osztályok (az adott projektben definiált osztályok)

Megjegyzés

- Az általunk használt objektumok (pl. Application, Debug) az Excel futásakor már létező (azaz hivatkozható) objektumok.
- Futásidőben is létrehozhatók új objektumok (pl. Worksheets.Add), illetve már létező objektumok megszüntethetők (pl. Charts(1).Delete).
- Az objektumtípusokat hasonlóan kezelhetjük, mint a VBA többi típusát (pl. `Dim r As Range`).

4.1.1. Az Application objektum

Az Application objektum az Excel alkalmazás objektuma. Segítségével az alkalmazáshoz tartozó összes objektum (pl. a nyitott munkafüzetek, projekt objektumok) elérhető. Az Application objektum feladata az alkalmazásszintű beállítások, tevékenységek végrehajtása is (pl. DisplayAlerts, ReferenceStyle).

Az Application objektum néhány fontosabb tulajdonsága: Workbooks, Worksheets, Charts, Sheets, ActiveWorkbook, ActiveSheet, ActiveCell, ActiveChart, Range, Cells, Rows, Columns, Selection.

Megjegyzés

- Az Application objektum tulajdonságaira, illetve metódusaira való hivatkozásból az Application szó elhagyható (pl. Application.ActiveCell helyett elegendő az ActiveCell hivatkozás), ezért általában el is hagyjuk. Az objektum nélküli hivatkozásokban (pl. Range) az Application objektum tulajdonságait, illetve metódusait hivatkozzuk.
- A Selection olyan speciális tulajdonság, amely az aktuálisan kijelölt objektumot adja eredményül, így az eredmény típusa attól függ, hogy éppen mi van kijelölve (pl. cellatartomány esetén egy Range objektum, diagram esetén egy Chart objektum lesz az eredmény).

4.1.2. Gyűjtemények

A gyűjtemények azonos típusú objektumok egy összessége. Egy gyűjtemény elemeire indexekkel (1-től induló sorszámokkal) hivatkozhatunk (pl. Workbooks(1).Worksheets(1).Cells(1,1)). Ha egy gyűjtemény elemei rendelkeznek név (Name) tulajdonsággal, akkor az elemekre a nevükkel is hivatkozhatunk (pl. Workbooks("Munkafüzet1.xlsm").Worksheets("Munka1").Cells(1,1)).

A gyűjtemények rendelkeznek Count tulajdonsággal, amivel a gyűjtemény elemeinek száma kapható meg, és többnyire Add metódussal is, amivel a gyűjtemény egy újabb elemmel bővíthető. A további tulajdonságok és metódusok (pl. az elemek törlésére szolgáló Delete metódus) létezése az egyes gyűjteményektől függ (pl. a Charts gyűjteménynek van ilyen metódusa, a Workbooks gyűjteménynek nincs).

A fontosabb gyűjtemények:

- Munkafüzetek: Workbooks gyűjtemény (munkafüzet (Workbook) objektumokból).
- Munkalapok: Worksheets gyűjtemény (munkalap (Worksheet) objektumokból).
- Diagramok: Charts gyűjtemény (diagram (Chart) objektumokból).
- Lapok (munkalapok, önálló lapon lévő diagramok, stb.): Sheets gyűjtemény.
- Párbeszédablakok: Dialogs gyűjtemény (beépített párbeszédablak (Dialog) objektumokból).
- Menük: CommandBars gyűjtemény (menüsor (CommandBar) objektumokból).

Megjegyzés

- A gyűjteményeket könnyű felismerni a nevük végén lévő „s" betűről (angol többes szám).
- A Sheets gyűjtemény elemeinek (lapjainak) típusa a Type tulajdonsággal kapható meg (Pl. Sheets(1).Type).
- Az önálló lapon lévő diagramok a Charts gyűjteménnyel, míg az egyes munkalapokon lévő diagramok az adott munkalap ChartObjects gyűjteményével (pl. Worksheets(1).ChartObjects) kezelhetők.

4.1.3. A Range objektum

A Range objektum cellatartomány kezelésére használatos. Az alábbiakban felsoroljuk a Range objektum néhány fontosabb tulajdonságát és metódusát, majd mintapéldákkal szemléltetjük ezek használatát.

Tulajdonságok: Address, Areas, Cells, Column, Columns, ColumnWidth, Count, CurrentRegion, Font, Formula, FormulaLocal, FormulaR1C1, FormulaR1C1Local, Height, Name, NumberFormat, NumberFormatLocal, Offset, Range, Resize, Row, RowHeight, Rows, Value, Width.

Metódusok: AutoFill, Clear, ClearFormats, Copy, Delete, Find, PasteSpecial, Select, Sort.

Megjegyzés: A cellatartományra a blokk szó is használatos (mert rövidebb), de a két dolog nem pontosan ugyanazt jelenti. Blokkon ugyanis egy összefüggő, téglalap alakú cellatartományt értünk. Egy cellatartomány állhat több blokkból is, de ez fordítva nem igaz.

Hivatkozás

- Az *A1 stílusú* hivatkozás: a cellatartományt egy sztringben megadott kifejezéssel hivatkozzuk.
  Pl. Range("A1"), Range("c2"), Range("A2:B3"), Range("A:A"), Range("1:1"), Range("A1:A5,C1:C5"), Range("A:A,C:D").
- Az *R1C1 stílusú* hivatkozás: a cellatartományt a sor- és oszlopindexek segítségével hivatkozzuk.
  Pl. Cells(1,1), Cells(2,3), Range(Cells(2,1), Cells(3,2)), Columns(1), Rows(1).

Kijelölés: Select metódus

Pl.

```
'Az első munkafüzet első munkalapján az A1:B3 blokk kijelölése
Application.Workbooks(1).Worksheets(1).Range("A1:B3").Select
'Az aktuális munkalap A1:B3 blokkjának kijelölése
Range("b3:a1").Select
```

Adatmegadás: Value tulajdonság

Pl.

```
'Az aktuális munkalap celláiba teszünk adatokat
Range("A1").Value = "Maci Laci"          'Egy szöveg
Range("A2").Value = 2*3                  'Egy szám
Cells(3,1).Value = Date                  'Az aktuális dátum
Cells(4,1).Value = CDate("2012.12.24")   'Egy adott dátum
Cells(4,1).Value = ""                    'Adat törlése
```

Megjegyzés

- A Value tulajdonság el is hagyható (pl. `Range("A2") = 3.14`)
- Egy cella üressége az IsEmpty függvénnyel kérdezhető le.

Tartalom törlése: ClearContents metódus
Pl.

```
'Az aktuális munkalap egy blokkjának tartalmát töröljük
Range("A1:A4").ClearContents
```

Formátum törlése: ClearFormats metódus
Pl.

```
'Az aktuális munkalap egy blokkjának formátumát töröljük
Range("A1:A4").ClearFormats
```

A tartalom és a formátum törlése: Clear metódus
Pl.

```
'Az aktuális munkalap egy blokkjának formátumát és tartalmát töröljük
Range("A1:A4").Clear
```

A blokk celláinak törlése: Delete metódus
Pl.

```
'Az aktuális munkalap egy blokkjának celláit töröljük
Range("A1:A4").Delete         'A jobbra lévő cellák balra lépnek
Range("A1:D1").Delete         'A lenti cellák feljebb lépnek
```

Megjegyzés: A Delete metódusnak van egy opcionális paramétere, amellyel megadható, hogy a törölt cellák helyére hogyan lépjenek be (jobbról vagy lentről) a szomszédos cellák. Ha a paramétert nem adjuk meg (mint a példában), akkor ezt a törölt objektum alakja alapján dönti el az Excel.

Formátum beállítása: NumberFormat, NumberFormatLocal tulajdonság
Pl.

```
'Az aktuális munkalap A oszlopára dátum formátumot
Range("A:A").NumberFormat = "yyyy.mm.dd"
'Ez ugyanazt csinálja, mint az előző sor
Columns(1).NumberFormatLocal = "éééé.hh.nn"
'Az aktuális munkalap C és D oszlopára pénznem formátumot
Range("C:D").NumberFormat = "#,##0 $"
```

Az aktuális cellát tartalmazó összefüggő blokk: CurrentRegion tulajdonság
Pl.

```
'Az aktuális munkalap A1-es celláját tartalmazó blokk méretei
s = Range("A1").CurrentRegion.Rows.Count       'Sorok száma
o = Range("A1").CurrentRegion.Columns.Count    'Oszlopok száma
```

Amint azt láttuk, az Excel munkalapokon tárolt adatok a cellák Value tulajdonságával elérhetők, azaz adatokat tudunk a cellákba tenni, és azokat fel tudjuk használni. Ha az adatokból valamilyen eredményadatokat szeretnénk meghatározni, akkor ezt általában nem programozzuk (pl. egy összeg esetén egy összegző algoritmussal), hanem az Excel-t „kérjük meg" az eredmények kiszámítására (pl. használjuk a SZUM függvényt). Ennek két oka is van. Egyrészt az Excel-ben nagyon sok beépített függvény használható, másrészt ez a megoldás követi az adatok esetleges megváltozását (ha a képletek automatikus számítása (Fájl, Beállítások, Képletek, Számítási beállítások, Munkafüzet kiszámítása, Automatikus

választógomb) be van kapcsolva, de ez általában bekapcsolt állapotú az Excel-ben). A programmal kiszámolt eredmények „frissüléséhez" ugyanis az eredményt számító forráskódot újra kell futtatni.

Ahhoz, hogy forráskódból olyan Excel képleteket tudjunk az egyes cellákba tenni, amelyek a kívánt feladatot elvégzik (kiszámolják a megfelelő eredményeket a megfelelően hivatkozott cellatartományok adataiból), szükségünk van az Excel cellahivatkozásainak ismeretére.

A cellahivatkozások stílusa az Excel-ben beállítható (lásd Fájl, Beállítások, Képletek, S1O1 hivatkozási stílus jelölőnégyzet). Általánosan elterjedt az *A1 stílusú* cellahivatkozás, amely az oszlopokat egy vagy több betűvel, míg a sorokat sorszámokkal jelöli. Beállítható azonban az ún. *S1O1 stílusú* cellahivatkozás is, ahol az oszlopok is sorszámokkal azonosíthatók.

Megjegyzés: Az S1O1 stílusú hivatkozásban az S betű a sort, az O betű az oszlopot jelenti. Az S1O1 cellahivatkozási stílus angol elnevezésében (R1C1 style) R jelenti a sort (row), C pedig az oszlopot (column).

A VBA környezetben az Excel cellahivatkozási stílusát az Application objektum ReferenceStyle tulajdonsága tárolja, így a cellahivatkozási stílus lekérdezése, illetve beállítása ezzel a tulajdonsággal történhet.

Pl.

```
'Lekérdezés
If Application.ReferenceStyle = xlR1C1 Then
  MsgBox ("Az Excel S1O1 stílusú cellahivatkozást használ")
Else
  MsgBox ("Az Excel A1 stílusú cellahivatkozást használ")
End If
'Beállítás
Application.ReferenceStyle = xlA1
```

Az S1O1 cellahivatkozási stílusban a cellákra a sorok és oszlopok sorszámával hivatkozunk. A cellahivatkozások ugyanúgy lehetnek abszolút, relatív és vegyes hivatkozások, mint az A1 stílus esetén. A relatív hivatkozásnál a megadott sorszámokat szögletes zárójelbe kell tenni, az abszolút hivatkozásnál nem. A relatív hivatkozás mindig a hivatkozást tartalmazó cellához képest relatív, ahhoz viszonyítva értendő. Ha a hivatkozást tartalmazó cella sorát, illetve oszlopát szeretnénk hivatkozni, akkor nem kell sorszámot megadnunk.

Az alábbiakban az S1O1 cellahivatkozási stílusokra adunk példákat (ahol a ~ karakter után megadjuk az S1O1 stílusú cellahivatkozás A1 stílusú megfelelőjét).

- Abszolút hivatkozás (pl. S1O1 ~ $A$1; S12O3 ~ $C$12).
- Relatív hivatkozás (pl. SO[1] ~ ugyanaz a sor, az oszlop az eggyel jobbra lévő; S[-1]O ~ a sor az eggyel feljebb lévő, az oszlop ugyanaz).
- Vegyes hivatkozás (pl. S2O[1] ~ abszolút sor (a második), relatív oszlop (az eggyel jobbra lévő); SO1 ~ relatív sor (ugyanaz a sor), abszolút oszlop (az első)).

Egy cellatartomány hivatkozásának lekérdezése: Address tulajdonság

Az Address tulajdonság használatának (egyszerűsített) szintaktikája:

`expression.Address(`*`RowAbsolute`*`, `*`ColumnAbsolute`*`, `*`ReferenceStyle`*`, `*`RelativeTo`*`)`

`RowAbsolute`	Az eredményhivatkozás sor abszolút legyen-e (alapértelmezésben `True`).
`ColumnAbsolute`	Az eredményhivatkozás oszlop abszolút legyen-e (alapértelmezésben `True`).
`ReferenceStyle`	Az eredményhivatkozás stílusa (`xlA1` vagy `xlR1C1`, alapértelmezésben `xlA1`).
`RelativeTo`	Az R1C1 típusú eredményhivatkozás viszonyítási Range objektuma.

Pl.

```
'Az aktuális munkalap egy cellájának hivatkozása A1 stílusban
MsgBox Cells(2,3).Address                          '$C$2
MsgBox Cells(2,3).Address(True, False)             'C$2
MsgBox Cells(2,3).Address(False, True)             '$C2
MsgBox Cells(2,3).Address(False, False)            'C2

'Az aktuális munkalap egy cellájának címe R1C1 stílusban
MsgBox Cells(2,3).Address(ReferenceStyle:=xlR1C1)          'R2C3
MsgBox Cells(2,3).Address(ReferenceStyle:=xlR1C1, _
  RowAbsolute:=False, ColumnAbsolute:=False, _
  RelativeTo:=Cells(1,1))                                  'R[1]C[2]
```

Megjegyzés: Ha egy utasítást több sorba írunk, akkor az alulvonás karaktert (_) kell azon sorok végére tenni, amelyek még folytatódnak (lásd a fenti példában).

Képletek használata: Formula, FormulaLocal tulajdonságok

Mindkét tulajdonság képlet megadására használatos. A képletet egy sztringkifejezéssel definiálhatjuk, lényegében ugyanúgy, mint ahogy azt az Excel-ben is megadnánk.

A Formula tulajdonságban az Excel függvényeire az angol nevükkel kell hivatkozni (az Excel súgója tartalmazza a függvények angol megnevezését és szintaxisát is), a FormulaLocal tulajdonság esetén pedig azon a nyelven, amit az Excel-ben is használunk.

A képletek könnyen megadhatók abban az esetben, ha tudjuk, hogy melyik cellába akarjuk a képletet elhelyezni, és ha az alkalmazni kívánt képletben ismerjük az esetlegesen hivatkozott cellatartományt. Nehezebb a dolgunk, ha nem tudjuk a képletet tartalmazó cella címét akkor, amikor a forráskódot írjuk (mert pl. ez függ a munkalapon lévő adatsorok, adatoszlopok számától).

A következőkben mindkét esetre mutatunk példát. Az első két utasítás ismert célcellába ismert cellatartományra hivatkozó képletet tesz. A másik két képlet elhelyezésekor nem tudjuk az adatokat tartalmazó blokk méretét, csak azt, hogy az A1-es cellától kezdődik az az összefüggő blokk, amelyre ki akarunk számolni valamit (példánkban átlagot és összeget számolunk). A CurrentRegion tulajdonság segítségével megtudható az adatokat tartalmazó blokk mérete. Az Address tulajdonság segítségével megkaphatók azok a (relatív) cellacímek, amelyek a képletekhez (az első oszlop átlagához, és az első sor összegéhez) kellenek.

Pl.

```
'Az aktuális munkalapon fix képletet fix helyre
Range("C5").Formula = "=SUM(C2:C4)"
Range("D5").FormulaLocal = "=SZUM(D2:D4)"
'Az A1-es cellát tartalmazó blokk alá az első oszlopba
s = Range("A1").CurrentRegion.Rows.Count
st = Cells(s, 1).Address(False, False)
Cells(s + 1, 1).FormulaLocal = "=ÁTLAG(A1:" + st + ")"
'Az A1-es cellát tartalmazó blokk mellé az első sorba
o = Range("A1").CurrentRegion.Columns.Count
st = Cells(1, o).Address(False, False)
Cells(1, o + 1).FormulaLocal = "=SZUM(A1:" + st + ")"
```

Képletek használata: FormulaR1C1, FormulaR1C1Local tulajdonságok

Ez a két tulajdonság olyan képletek megadására használható, amelyekben a cellatartományokra az R1C1 cellahivatkozási stílussal hivatkozunk. A következő példák mindegyike a C2-es cellába helyez el egy képletet, amely egy cella tartalmának kétszeresét számolja ki. A cella, amire hivatkozunk, a hivatkozástól függően változik. A C2-es cellába kerülő A1 hivatkozási stílusú képletet a sorok végén található megjegyzésekben láthatjuk.

Pl.

```
'Egy egyszerű képletet a C2-es cellába
Range("C2").FormulaR1C1 = "=R1C1*2"             ' "=$A$1*2"
Range("C2").FormulaR1C1 = "=RC1*2"              ' "=$A2*2"
Range("C2").FormulaR1C1 = "=R1C*2"              ' "=C$1*2"
Range("C2").FormulaR1C1 = "=R[1]C[-1]*2"        ' "=B3*2"
'A FormulaR1C1Local tulajdonságban az S és O betűk szerepelnek
Range("C2").FormulaR1C1Local = "=S1O1*2"        ' "=$A$1*2"
Range("C2").FormulaR1C1Local = "=S[1]O[-1]*2"   ' "=B3*2"
```

Megjegyzés

- A példákban most nem használtuk az Address tulajdonságot, de a hivatkozott cella R1C1 stílusú címe ezzel is előállítható lett volna.
- Az Excel-ben a függvénynevek magyarul (helyi (local) nyelven) jelennek meg (akkor is, ha a Formula, illetve FormulaR1C1 tulajdonságokkal definiáljuk őket).
- A képleteket kezelő tulajdonságok (Formula, FormulaLocal, FormulaR1C1, FormulaR1C1Local) egyikének megadásával mindegyik definiálódik.

  Pl.

  ```
  Range("C2").FormulaLocal = "=ÁTLAG(A2:B2)"
  MsgBox Range("C2").FormulaR1C1            ' =AVERAGE(RC[-2]:RC[-1]
  ```

Képletek másolása: Copy, AutoFill, PasteSpecial metódusok

A képleteket tartalmazó cellákat gyakran másoljuk, hogy ne kelljen azokat többször megadni. A másolás az Excel-ben többféle módon is elvégezhető (pl. a vágóasztal segítségével, a másolandó cella jobb alsó sarkánál lévő kitöltőjel segítségével), így ezt forráskódból is többféleképpen végezhetjük.

Pl.

```
'Az s+1. sor 1. oszlopában lévő képlet másolása az s+1. sor
'oszlopaiba a 2. oszloptól az o. oszlopig
Cells(s + 1, 1).Copy Destination:= _
  Range(Cells(s + 1, 2), Cells(s + 1, o))
'Az 1. sor o+1. oszlopában lévő képlet másolása az o+1. oszlop
'soraiba a 2. sortól az s. sorig
'AutoFill esetén a céltartománynak a forrást is tartalmaznia kell
Cells(1, o + 1).AutoFill Destination:= _
  Range(Cells(1, o + 1), Cells(s, o + 1))
'Mint az előző, csak másolással és beillesztéssel
Cells(1, o + 1).Copy
'Beillesztés
Range(Cells(2, o + 1), Cells(s, o + 1)).PasteSpecial
'A forrástartományt jelölő (villogó) szegély levétele
Application.CutCopyMode = False
```

Egyéb lehetőségek: Rows, Columns, Cells, Range, Offset, Resize

A Range objektum fenti tulajdonságai Range objektumot adnak eredményül. Az első négy tulajdonság jelentése és használata értelemszerű; az Offset tulajdonsággal egy, az adott Range objektumhoz képest relatív elmozdulással nyert Range objektum hivatkozható; a Resize tulajdonsággal pedig átméretezhető egy adott blokk. A példákban az egyes tulajdonságok eredményeként kapott Range objektumoknak a Select metódusát hívjuk meg (amellyel látható lesz az eredmény). Az utolsó példa az A1-es cella egy feleslegesen cifra hivatkozását szemlélteti.

Pl.

```
'Egy Range objektum egy sora, illetve egy oszlopa
Range("C3:D6").Rows(2).Select
Range("C3:D6").Columns(2).Select
'Egy Range objektum egy cellája
Range("C3:D6").Cells(1, 2).Select
'Ugyanazt jelöli ki, mint az előző
Range(Cells(3, 3), Cells(6, 4)).Range("B1").Select
'Egy Range objektum egy Range objektuma
Range("C3:D6").Range("A2:B2").Select
'Egy Range objektumból relatív elmozdulással nyert Range objektum
Range("C3:D6").Offset(-1, 1).Select
'Egy Range objektum átméretezése
Cells(2,2).Resize(3,2).Select
'Range objektumok egyesítése
Application.Union(Columns(1), Columns(4)).Select
'Az A1-es cella egy feleslegesen cifra hivatkozása
Range("A1").Cells(1,1).Offset(0,0).Resize(1,1).Cells(1).Select
```

4.1.4. A WorksheetFunction objektum

Az előző fejezetben láttuk, hogyan helyezhetünk el képleteket az egyes cellákba. Ezzel a megoldással az Excel végzi a számolásokat, és jeleníti meg az eredményeket a képleteket tartalmazó cellákban. Ennél a megoldásnál tehát bizonyos cellák tartalma módosul. Előfordulhat azonban olyan eset, amikor ez a megoldás nehezebben realizálható (pl. ha az adatokat tartalmazó munkalap védett, akkor egy másik (nem védett, akár egy másik munkafüzethez tartozó) munkalap szükséges a megoldáshoz).

Az Excel munkalapfüggvényei azonban nemcsak a cellákban elhelyezett képletekben használhatók. A forráskódból meghívható munkalapfüggvényeket a WorkSheetFunction tároló (container) objektum foglalja egy logikai egységbe. Az objektum függvénymetódusai az egyes munkalapfüggvények, amelyek paraméterei egyaránt lehetnek munkalapok Range objektumai, és memóriaváltozók (pl. egy tömbváltozó).

A függvénymetódusok hasonlóan hívhatók, mint a VB függvényei (a hívás bárhol állhat, ahol az eredmény típusának megfelelő érték állhat), így az eredmény megkapható a cellák módosítása nélkül is (igaz, az adatok módosulása esetén a számolást végző forráskódot újra kell futtatni).

A forráskódban az objektumtípusok hasonlóan használhatók, mint a VB többi adattípusa. Az objektumok értékadó utasítása a `Set` (nem opcionális) kulcsszóval kezdődik (szemben az eddig használt értékadás `Let` kulcsszavával, ami elhagyható volt).

Az objektumok értékadó utasításának (egyszerűsített) szintaktikája:

```
Set objectvar = objectexpression
```

Az értéket kapó objektumváltozónak (*objectvar*) ugyanolyan típusúnak kell lennie, mint az objektumkifejezés (*objectexpression*) eredményeként előálló objektumnak.

Az alábbi példa a WorksheetFunction objektum használata mellett az objektumok értékadását is bemutatja.

Pl.

```
'Munkalapfüggvény használata cellatartományra
Sub MunkalapFgv1()
Dim r As Range, min As Variant
Set r = Application.Worksheets("Munka1").Range("A1:C5")
min = Application.WorkSheetFunction.Min(r)
MsgBox min
End Sub

'Munkalapfüggvény használata változókra
Sub MunkalapFgv2()
Dim a(1 To 10) As Integer, i As Integer
For i = 1 To 10: a(i) = i: Next
MsgBox WorkSheetFunction.Sum(a)     '55
End Sub
```

Megjegyzés

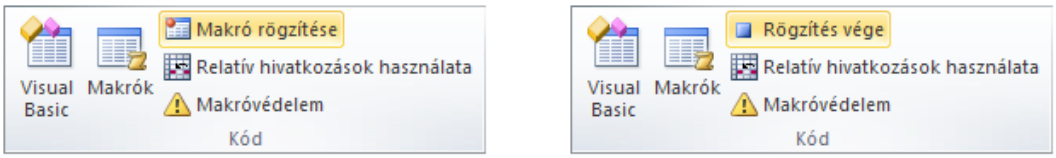
- Az egyes munkalapokra kérhető olyan beállítás is, amely a képleteket tartalmazó cellákban magát a képletet és nem az eredmény értékét jeleníti meg (lásd Fájl, Beállítások, Speciális, Beállítások megjelenítése ehhez a munkalaphoz, Számított eredmények helyett képletek megjelenítése a cellákban jelölőnégyzet).
- Általában akkor használunk külön objektumváltozót (a példában `r`), ha az adott objektummal több dolgot is el szeretnénk végezni. A példában csak a Min munkalapfüggvényt hívjuk meg egy adott blokkra, ami az objektumváltozó nélkül is megtehető lett volna.
- A példában szereplő `min` változó `Variant` típusú, mert az eredmény típusa a megfelelő cellákban (A1:A5) található adatok típusától függ.

4.2. Egyéb VBA lehetőségek

Ebben a fejezetben olyan témákkal foglalkozunk, amelyek talán nem alapvető fontosságúak az Excel programozását illetően, de elég hasznosak ahhoz, hogy ha röviden is, de érintsük őket.

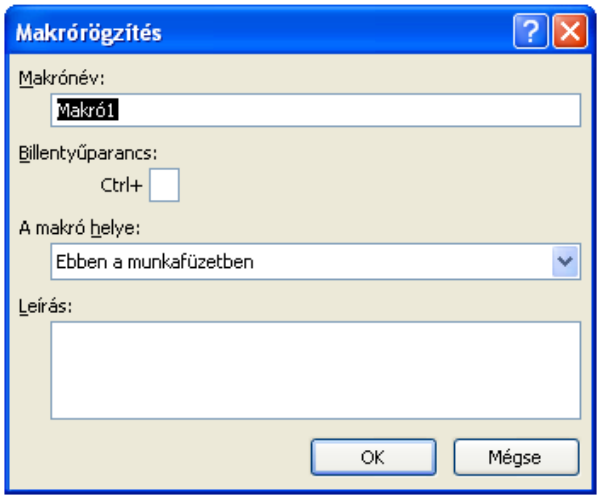
4.2.1. Makrók rögzítése

Az Excel képes arra, hogy a felhasználó által elvégzett tevékenységeket rögzítse, azaz forráskód szinten megadja azokat a VBA utasításokat, amelyekkel az elvégzett tevékenységek végrehajthatók. A Makró rögzítése funkció az Excel menüszalagján a Fejlesztő eszközök lap Kód csoportjában található (ott, ahol a Visual Basic Editor program ikonja is szerepel, lásd 4.1. ábra).



4.1. ábra. A Makró rögzítése és a Rögzítés vége funkciók a menüszalag Fejlesztőeszközök lapján

A Makró rögzítése funkció egy párbeszédablakot jelenít meg (lásd 4.2. ábra), ahol megadhatók a rögzítendő makró adatai (név, gyorsbillentyű, hely, leírás).



4.2. ábra. A rögzítendő makró adatainak megadására szolgáló párbeszédablak

A rögzített makrók egy új modulba kerülnek, a forráskódok tetszőlegesen felhasználhatók (szerkeszthetők, futtathatók, stb.). Az alábbi példa egy olyan rögzített makró forráskódját tartalmazza, amelyben egy cella tartalmát töröltük.

Pl.

```
Sub Makró1()
'
' Makró1 Makró
'
'

'
    Range("B4").Select
    Selection.ClearContents
End Sub
```

4.2.2. Diagramok kezelése

Az Excel-ben könnyen készíthetünk adatainkból szemléletes diagramokat. A diagramok kezelése (létrehozás, módosítás, törlés, stb.) VBA utasításokkal is elvégezhető.

A diagramkészítés tipikus lépéseit egy példa segítségével szemléltetjük. A diagramot az első munkalap A1-es celláját tartalmazó összefüggő blokk adataiból készítjük, és az első munkalapon helyezzük el. Először a Charts gyűjtemény Add metódusával létrehozunk egy új (még üres) diagramot, majd megadjuk a diagram legfontosabb adatait (típus, forrásadatok, célhely), végezetül pedig (hogy ne a diagram legyen az aktuálisan kijelölt objektum), az első munkalap A1-es celláját aktivizáljuk.

Pl.

```
'Diagramkészítés
Sub Diagram()
Dim r As Range
'Az A1-es cellát tartalmazó összefüggő blokk adataiból...
Set r = Worksheets(1).Range("A1").CurrentRegion
'Egy új, üres diagram létrehozása
Charts.Add
'A diagram testreszabása
With ActiveChart
  'A diagram típusa
  .ChartType = xlColumnClustered
  'A diagram forrásadatai és az adatsorozatok képzése (sorokból)
  .SetSourceData Source:=r, PlotBy:=xlRows
  'A diagram elhelyezése: objektumként az első munkalapra
  .Location Where:=xlLocationAsObject, Name:=Worksheets(1).Name
End With
Worksheets(1).Activate        'Az első munkalap aktivizálása
Range("A1").Select            'Az A1-es cella kijelölése
End Sub
```

A következő példa törli az önálló lapon, valamint az első munkalapon lévő diagramokat. Az önálló lapon lévő diagramok törlésekor (mivel teljes lapok törlődnek), az Excel egy figyelmeztető üzenetet ad, és a törlés csak jóváhagyás után történik meg. Ennek a figyelmeztető üzenetnek a megjelenítését kapcsoljuk ki az Application objektum DisplayAlerts tulajdonságával.

Pl.

```
'Diagramok törlése
Sub DiagramTorles()
'Az önálló lapon lévőket
If Charts.Count > 0 Then
  Application.DisplayAlerts = False       'Ne legyen figyelmeztető üzenet
  Charts.Delete
  Application.DisplayAlerts = True        'A figyelmeztetés visszakapcsolása
End If
'Az első munkalapon lévőket
If Worksheets(1).ChartObjects.Count > 0 Then
  Worksheets(1).ChartObjects.Delete
End If
End Sub
```

Megjegyzés

- A diagramokat tartalmazó gyűjtemények (Charts, ChartObjects) Delete metódusa futási hibát ad akkor, ha a gyűjteménynek nincs egyetlen eleme sem, ezért a törlést csak akkor végezzük el, ha van mit törölni.
- Az Application objektum DisplayAlerts tulajdonsága az összes figyelmeztető üzenet megjelenítését szabályozza (pl. egy nem mentett munkafüzet bezárásakor megjelenő üzenet sem jelenik meg, ha ez a kapcsoló ki van kapcsolva (`False` értékű), ezért a törlés után visszakapcsoljuk (`True` értékűre)).

4.2.3. A beépített párbeszédablakok használata

Az Excel sokféle párbeszédablakot használ, amelyek a VBA környezetből is elérhetők. Az Application objektum Dialogs gyűjteménye ezeket a párbeszédablakokat (Dialog objektumokat) foglalja egy logikai egységbe.

A forráskódból használni kívánt párbeszédablak a Dialogs gyűjtemény egy elemének a kiválasztásával hivatkozható. A kiválasztás egy XlBuiltInDialog felsorolt típus egy elemének a megadásával történhet (a példában a fájlmegnyitó párbeszédablakot hivatkozzuk az xlDialogOpen elemmel).

A párbeszédablakok Show metódusa megjeleníti a párbeszédablakot, ahol a felhasználó megadhatja a megfelelő adatokat. A metódus eredményül egy logikai értéket ad attól függően, hogy a felhasználó melyik nyomógombbal zárta be az ablakot. Az OK gomb esetén `True` az eredmény, a Mégse (Cancel) gomb (illetve az ablak bezárása) esetén `False`.

Az alábbi példa a fájlmegnyitó párbeszédablak segítségével megadott (Excel-ben megnyitható típusú) fájl teljes elérési útját írja ki a képernyőre. Az OK gomb választása esetén a Show metódus meg is nyitja a kiválasztott fájlt, így ez lesz az aktuális munkafüzet. A nyitott munkafüzetek (így az aktuális munkafüzet) teljes elérési útja a FullName tulajdonsággal érhető el. Ennek megjegyzése után bezárjuk a megnyitott fájlt, így újra a makrókat tartalmazó munkafüzet lesz az aktuális (aktív) munkafüzet.

Pl.

```
'Fájlválasztás párbeszédablakkal
Sub FajlValasztas()
Dim s As String
s = ""
If Application.Dialogs(xlDialogOpen).Show Then
  'A Megnyitás gombot nyomták meg, megjegyezzük a fájl specifikációját
  s = ActiveWorkbook.FullName
  'Lezárjuk a megnyitott munkafüzetet
  ActiveWorkbook.Close
End If
If s = "" Then
  MsgBox "Nem választottak fájlt!"
Else
  MsgBox "A kiválasztott fájl:" + vbCrLf + s
End If
End Sub
```

Megjegyzés

- A Dialogs gyűjtemény csak olvasható (read-only) (így pl. nincs Add metódusa sem).
- Ha a példában kiválasztunk egy fájlt, akkor egy kétsoros üzenetet kapunk, ahol a kiválasztott fájl teljes specifikációja (meghajtó, elérési út, fájlazonosító) a második sorba kerül. A sortörést a `vbCrLf` konstanssal végeztük (Chr(13)+Chr(10)).

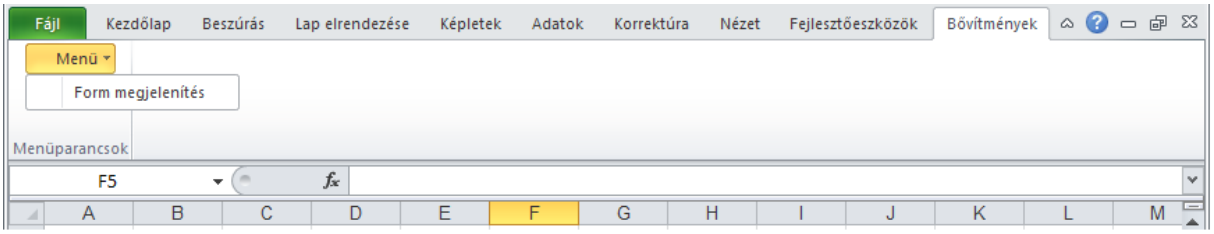
4.2.4. Saját menü készítése

Ha az Excel funkcióit saját fejlesztésű makrókkal bővítjük, akkor a makróink elérhetőségét, futtathatóságát egyszerű, a felhasználók által könnyen kezelhető módon kell biztosítanunk. Eddig csak olyan futtatási lehetőségekről volt szó, amelyhez a Visual Basic Editor-t is használni kellett. Ezt azonban egy átlagos Excel felhasználótól nem várhatjuk el.

Szerencsére megvannak azok az eszközök, amellyel ez a probléma megoldható. Egyfelől az Excel menüje VBA eszközökkel kezelhető, így pl. forráskódból bővíteni tudjuk az Excel menüjét a makróink futtatását végző funkciókkal. Másfelől a munkafüzetekhez is tartoznak eseménykezelők, amelyek automatikusan lefutnak a munkafüzethez kapcsolódó események (pl. megnyitás, mentés, bezárás) bekövetkezésekor.

A makróinkat tartalmazó munkafüzet megnyitásakor automatikusan (külön indítás nélkül) lefuttathatjuk az Excel menüjét bővítő makrónkat, így a többi makró (az általunk készített funkciók) már az Excel menüjéből indítható. A makróinkat tartalmazó dokumentum bezárásakor pedig visszaállíthatjuk a menü eredeti állapotát.

Az alábbi példa ezt szemlélteti. A `MenuKirak` szubrutin bővíti az Excel menüjét egy új (Menü feliratú) menüponttal. Ez a menüpont az Excel menüszalagján a Bővítmények lapon fog megjelenni (lásd 4.3. ábra). A menün belül egyetlen almenüpontot definiálunk (Form megjelenítés felirattal), ami egy formot (`UserForm1`) jelenít meg. A `MenuLevesz` szubrutin törli a `MenuKirak` által (akár több példányban) létrehozott Menü feliratú menüpontot.



4.3. ábra. A saját menü megjelenése a menüszalag Bővítmények lapján

Pl.

```
'Saját menü létrehozás
Sub MenuKirak()
Dim fomenu As CommandBar
Dim fomenupont As CommandBarControl, almenupont As CommandBarControl
Set fomenu = Application.CommandBars.ActiveMenuBar
'Egy új (felbukkanó) menüpont létrehozása
Set fomenupont = fomenu.Controls.Add(Type:=msoControlPopup)
fomenupont.Caption = "Menü"
'Az új főmenüponthoz egy új almenüpontot
Set almenupont = fomenupont.CommandBar.Controls.Add(Type:=msoControlButton)
almenupont.Caption = "Form megjelenítés"
'A menüpont aktivizálásakor lefutó szubrutin
almenupont.OnAction = "FormKirak"
End Sub

Sub FormKirak()
UserForm1.Show
End Sub
```

```
'Saját menü levétele
Sub MenuLevesz()
Dim menupont As CommandBarControl
For Each menupont In Application.CommandBars.ActiveMenuBar.Controls
  If menupont.Caption = "Menü" Then
    menupont.Delete
  End If
Next
End Sub

'Az eredeti rendszermenü visszaállítása (ha elrontanánk a menüt)
Sub MenuAlaphelyzet()
Application.CommandBars("Worksheet Menu Bar").Reset
End Sub
```

A fenti szubrutinokat (menükezelés, a menüpontok aktivizálásakor végrehajtandó szubrutinok) egy modulban helyezendők el (így elérhetők, futtathatók lesznek), míg az alábbi eseménykezelő szubrutinokat a ThisWorkbook objektum moduljában kell elhelyezni.

Pl.

```
'A munkafüzet megnyitásakor aktivizálódik
Private Sub Workbook_Open()
'A saját menü kirakása
Call MenuKirak
End Sub

'A munkafüzet lezárásakor aktivizálódik
Private Sub Workbook_BeforeClose(Cancel As Boolean)
'A saját menü levétele
Call MenuLevesz
End Sub
```

Megjegyzés

- A `MenuAlaphelyzet` szubrutint azért készítettük, hogy ha elrontanánk az Excel menüjét, akkor gyorsan visszaállítható legyen a menü alapállapota. A menüszalag alaphelyzetbe állítása ugyan az Excel beállításai között is megtalálható (lásd Fájl, Beállítások, Menüszalag testreszabása, Alaphelyzet), de ez nem érinti a forráskódból megtett változtatásokat.
- A példában csak kétféle típusú vezérlőt használtunk (`msoControlPopup`, `msoControlButton`), de másféle (pl. legördülő listát megvalósító) vezérlő is elhelyezhető a menüben.

4.3. Mintafeladat

A fejezetet (s egyben az egész témakört) a következő, összetettebb feladat megoldásával zárjuk. Igaz, hogy az Excel-t rendszerint adatfeldolgozási feladatokra használjuk, így a programozási feladatok zöme is ilyen jellegű, most mégis (kicsit kedvcsináló célzattal) egy másfajta feladatot választottunk. Bizonyára mindenki ismeri a Torpedó nevű játékot, amelyben két személy próbálja meg mielőbb „kilőni" az ellenfél által elrejtett alakzatokat.

A játéktér egy-egy (pl. 10×10-es) négyzetrács (egy az elrejtett, egy pedig a kilövendő alakzatokhoz), amelyben az oszlopokat angol betűkkel (pl. A-tól J-ig), a sorokat sorszámokkal (pl. 1-től 10-ig) azonosítjuk. Az egyes négyzeteket (ahová elrejteni, illetve lőni lehet) az oszlop- és sorazonosítók határozzák meg (pl. A1, C4). Az előre rögzített számú és formájú alakzat elrejtése után a két játékos felváltva lő, és a játékot az nyeri, aki előbb lövi ki (süllyeszti el) az ellenfél alakzatait.

A játék eredeti formában történő megvalósítása (két játékos, több négyzetből álló alakzatok) „túl nagy falat" lenne, ezért egy egyszerűsített változatot definiálunk és oldunk meg. Ez az egyszerűbb feladat is alkalmas lesz az eddig tanult ismeretek összefoglalására, és talán tartogat még annyi kihívást, illetve játékélményt, hogy megérje akár önállóan megoldani, akár „összerakni" és kipróbálni az általunk adott megoldást.

Feladat: Egy párbeszédablak segítségével valósítsuk meg az alábbi *Torpedó* játékot!

- A játék pályaméretét egy *Pályaméret* feliratú legördülő listával lehessen megadni, ami 2-től 6-ig egy egész szám, kezdőértéke legyen 4! A legördülő lista feltöltését ciklussal végezzük!
- A *pálya* az első munkalap A1-es cellájától kezdődő *pályaméret*pályaméret* méretű blokkja, míg az elrejtett alakzatok tárolására a második munkalap hasonló blokkját használjuk *(háttér)!* A játék során az első munkalapot lássuk!
- Egy *Elrejt* feliratú nyomógomb megnyomására helyezzünk el a *háttéren* véletlenszerűen annyi alakzatot, amennyi a *pályaméret!* Az alakzatok egy cellából állnak, jelzésükre az „O" betűt használjuk!
- Egy *Játék indít* feliratú nyomógomb megnyomására (ha már rejtettünk el alakzatokat) kezdjük el a játékot! A gomb felirata legyen *Játék leállít,* és amíg a játéknak nincs vége, addig ne lehessen pályaméretet választani és alakzatokat elrejteni, viszont lehessen az elrejtett alakzatokra lőni! Egy elkezdett játék a *Játék leállít* gomb megnyomásáig, vagy az összes elrejtett alakzat kilövéséig tart.
- A lövés célcellájának címe egy beviteli mezőben legyen megadható! A lövést egy *Lövés* feliratú (az Enter billentyűre aktív) nyomógombbal lehessen leadni! Csak pályára eső célhelyre lehessen lőni!
- Értékeljük ki a lövést úgy, hogy ha olyan helyre lőttünk, ahová már lőttünk, akkor erről tájékoztassunk, ha eltaláltunk egy alakzatot (a másik munkalapon elrejtett alakzatok közül), akkor azt jelezzük a *pálya* megfelelő cellájában elhelyezett „X" (talált) vagy „*" (nem talált) karakterrel!
- Egy *Súgó* feliratú nyomógomb megnyomására adjunk „súgót" az elrejtett alakzatokról, azaz jelenítsük meg a második munkalapot, hogy az elrejtett alakzatok helyét megnézhessük! Amíg a „súgót" látjuk, addig ne lehessen lövést leadni, valamint a súgó gomb felirata legyen *Bezár!* A *Bezár* gomb megnyomására állítsuk vissza a játék állapotot (azaz a *pályát* lássuk, *Súgó* gombfelirat, lövés engedélyezése)!
- Az ablakban jelenjen meg a kilőtt alakzatok száma! Ha az összes alakzatot kilőttük, akkor a játéknak vége, erről tájékoztassunk, majd kezdhessünk új játékot!

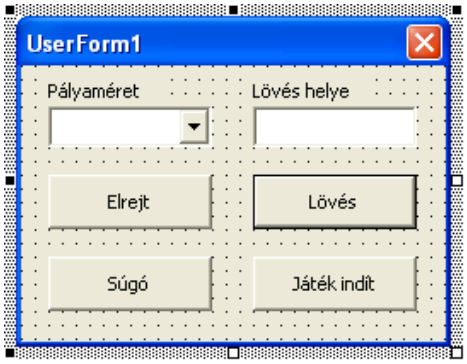
Megoldás: A feladat megoldását az egyes részfeladatokhoz illeszkedően, több lépésben végezzük.

1. lépés: A játék párbeszédablakának (formjának) elkészítése.

A form létrehozása után elhelyezzük a megfelelő vezérlőket a formon: egy legördülő listát (ComboBox), egy beviteli mezőt (TextBox), két címkét (Label), és négy darab nyomógombot (CommandButton). A két címkevezérlő a legördülő lista és a beviteli mező feladatáról tájékoztat, ezért feliratukat (Caption) *Pályaméret,* illetve *Lövés helye* értékekre állítjuk.

A forráskód könnyebb olvashatósága érdekében a vezérlőknek új nevet (Name) adunk. A legördülő lista nevét *cbPalya,* a beviteli mező nevét *tbLoves,* a négy nyomógomb nevét pedig *btnElrejt, btnLoves, btnSugo, btnJatek* értékekre állítjuk. A nyomógombok feliratait is megadjuk: *Elrejt, Lövés, Súgó, Játék indít.*

Ellenőrizzük a vezérlők (Tab Order) sorrendjét, és ha nem megfelelő, beállítjuk az elrendezéshez igazodóan.



4.4. ábra. A játék formja tervezéskor

Megjegyzés

- A vezérlők neveit a típusukra utaló rövidítésből és a funkciójukra utaló névből állítottuk össze (ami egyfajta programozási konvenció).
- A két címkevezérlőnek és magának a formnak nem adtunk külön nevet, mert nem hivatkozunk ezekre a forráskódból. Ha egy több formos alkalmazást készítünk, akkor a formnak is célszerű beszédesebb nevet adni.
- A pályaméret legördülő lista tartalmát (2-től 6-ig, egész számok) tervezéskor nem tudjuk beállítani, ezért a lista feltöltését futáskor (a kezdőtevékenységek között) kell majd elvégeznünk.
- A legördülő lista egy tulajdonsága (Style) azt szabályozza, hogy meg lehet-e adni új adatot (fmStyleDropDownCombo), vagy csak a legördülő listában szereplő elemekből lehet választani (fmStyleDropDownList). A feladathoz az utóbbi illeszkedik, ezért ezt fogjuk használni (amit szintén futási időben állítunk majd be).
- A párbeszédablakokban (a gyorsabb kezelés érdekében) gyakran használunk az Enter, illetve az Esc billentyű leütésére aktivizálódó nyomógombokat. Ehhez a megfelelő nyomógombok Default, illetve Cancel tulajdonságát kell igazra (`True`) állítani. A példánkban a Lövés feliratú gomb Default tulajdonságát állítottuk igazra, így a lövés célcellájának megadása után elegendő leütnünk az Enter billentyűt a lövés végrehajtásához (amit a Lövés feliratú nyomógomb végez). A Default nyomógomb szélesebb kerettel emelődik ki mind tervezéskor (lásd 4.4. ábra), mind futáskor (lásd 4.5. ábra).

2. lépés: Modulszintű utasítások megadása.

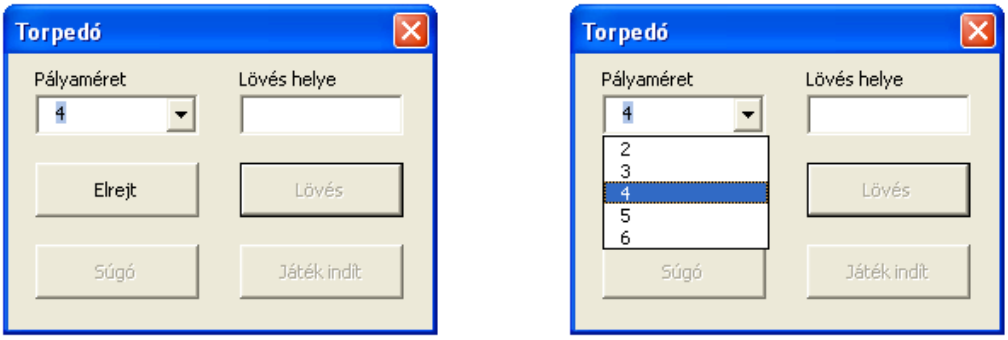
Az alábbi modulszintű utasítások a form moduljának elején helyezendők el. Használjuk a változók kötelező deklarálását kikényszerítő (`Option Explicit`) utasítást, majd a feladathoz illeszkedő konstansokat deklaráljuk (pl. a pályaméret határait), illetve azokat a modulszintű változókat, amelyeket több szubrutinból is hivatkozni szeretnénk (pl. az aktuális pályaméret).

```
'A változók kötelező deklarálásához
Option Explicit
'A pályaméret határai
Const Tol = 2
Const Ig = 6
'A lövések jelzésére
Const Talalt = "X"
Const Melle = "*"
Const Alakzat = "O"
'A form fejlécéhez
Const Fejlec = "Torpedó"
'Az aktuális pályaméret
Dim n As Integer
'A kilőtt alakzatok száma
Dim Kilott As Integer
```

3. lépés: Kezdőtevékenységek.

A form megjelenése előtti kezdőtevékenységeket a form Initialize eseménykezelőjében kell programozni. Feltöltjük a legördülő lista tartalmát (az AddItem metódus segítségével, a feladatkiírás szerint ciklussal), majd beállítjuk a legördülő lista kezdőértékét (a Value tulajdonsággal).

```
'A form megjelenése előtt fut le
Private Sub UserForm_Initialize()
Dim i As Integer
'A legördülő lista feltöltése
For i = Tol To Ig
  cbPalya.AddItem i
Next
'A legördülő lista kezdőértéke és stílusa
cbPalya.Value = 4: cbPalya.Style = fmStyleDropDownList
'Az ablak fejlécének beállítása
Caption = Fejlec
'A vezérlők választhatóságának beállítása
btnSugo.Enabled = False: btnJatek.Enabled = False
btnLoves.Enabled = False: tbLoves.Enabled = False
'Az első munkalapot lássuk
Worksheets(1).Activate
'Pálya törlése
Call Torles
'A véletlenszám-generátor inicializálása
Randomize
End Sub
```



4.5. ábra. A játék formja kezdetben, és a pályaméret választás

Megjegyzés

- A legördülő listák (és a listák (ListBox)) elemei sztring típusúak. Az AddItem metódust most egész számokkal hívjuk meg, ezért egy implicit típuskonverzió történik (a szám sztringgé alakul), és ezután kerül az új elem a legördülő lista elemei közé. Az új elemek alapértelmezetten a lista végére kerülnek (ahogy most is), de az AddItem meghívható adott helyre történő beszúrásra is.
- A pálya törlését elvégző segédszubrutint (`Torles`) is meghívjuk (amelynek kifejtése a 4. lépésnél található).
- A Randomize utasítás a véletlenszám-generátor inicializálását végzi el. Ekkor a véletlen számok előállítására használt Rnd függvény (lásd 5. lépés) futásonként eltérő véletlen számokat fog előállítani.

4. lépés: A pályaméret változása, a játéktér törlése, a kilőtt alakzatok számának megjelenítése.

Ebben a lépésben a pályaméret megváltozását lekezelő eseménykezelő, és két segédszubrutin kapott helyet. A pályaméret megváltozásakor (`cbPalya_Change`) az aktuális méretet megjegyezzük (az `n` változóban), a `Torles` szubrutin a maximális területű pályát törli (hiszen nagyobb pálya után egy kisebb pályán játszva, ha csak az aktuális méretű pályát törölnénk, akkor esetleg ott maradna egy-egy zavaró karakter az előző játékból), a harmadik pedig a form fejlécét (Caption) aktualizálja.

```
'A legördülő lista értékének megváltozásakor fut le
Private Sub cbPalya_Change()
n = cbPalya.Value
End Sub

'A játéktér (pálya, háttér) törlése az aktuális munkalapon
Sub Torles()
Range(Cells(1, 1), Cells(Ig, Ig)).Clear
End Sub

'A kilőtt alakzatok számát a form fejlécében jelenítjük meg
Sub FejlecKirak()
Caption = Fejlec + " (kilőve:" & Kilott & "db)"
End Sub
```

Megjegyzés: Az aktuális pályaméret beállításakor egy implicit típuskonverzió történik, mivel a legördülő listában lévő elemek sztringek (amelyek most egész számokat tartalmaznak (lásd 3. lépés), így a konverzió végrehajtható), az `n` változó pedig egész (`Integer`) típusú.

5. lépés: Alakzatok elrejtése.

Az alakzatok elrejtésénél `n` db alakzatot kell elhelyezni egy `n×n`-es táblázatban úgy, hogy nem teszünk ugyanarra a helyre több alakzatot. Az elrejtett alakzatokat a második munkalapra kell elhelyezni, ezért először aktivizáljuk a második munkalapot. Ezután töröljük az esetleg kint lévő, korábban elrejtett alakzatokat (`Torles`). A külső ciklus (`For`) biztosítja az `n` db alakzat elrejtését, a belső ciklus (`Do`) pedig azt, hogy egy véletlenszerű, még üres cellába (IsEmpty) kerüljön az éppen elrejtendő alakzat. Végezetül az első munkalapot aktivizáljuk, és választhatóvá tesszük a játék indítására szolgáló nyomógombot.

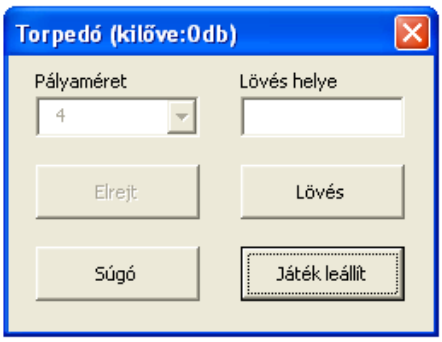
```
'Alakzatok (n db) véletlenszerű elrejtése
Private Sub btnElrejt_Click()
Dim i As Integer, s As Integer, o As Integer
Worksheets(2).Activate
Call Torles
For i = 1 To n
  Do
    s = Int(Rnd * n) + 1
    o = Int(Rnd * n) + 1
  Loop Until IsEmpty(Cells(s, o))
  Cells(s, o).Value = Alakzat
Next
Worksheets(1).Activate
'A játékot elkezdhessük
btnJatek.Enabled = True
End Sub
```

4.6. ábra. Elrejtés után már indítható a játék

6. lépés: Játék indítása, leállítása.

Itt lényegében csak a megfelelő vezérlők választhatóságát állítjuk be attól függően, hogy a játékot éppen indítjuk vagy leállítjuk. Azt, hogy éppen mit kell tenni, azt a nyomógomb aktuális feliratából deríthetjük ki. A játék leállításakor az elrejtésre szolgáló nyomógombot (`btnElrejt`) hozzuk fókuszba.

```
'Játék indítása, leállítása
Private Sub btnJatek_Click()
If btnJatek.Caption = "Játék indít" Then
  'A pálya törlése
  Call Torles
  btnJatek.Caption = "Játék leállít"
  btnLoves.Enabled = True: tbLoves.Enabled = True
  btnSugo.Enabled = True
  cbPalya.Enabled = False: btnElrejt.Enabled = False
  Kilott = 0
  Call FejlecKirak
Else
  btnJatek.Caption = "Játék indít"
  btnJatek.Enabled = False: btnLoves.Enabled = False
  tbLoves.Enabled = False: btnSugo.Enabled = False
  cbPalya.Enabled = True: btnElrejt.Enabled = True
  'Az Elrejt gomb legyen kiválasztva
  btnElrejt.SetFocus
  Caption = Fejlec
End If
End Sub
```

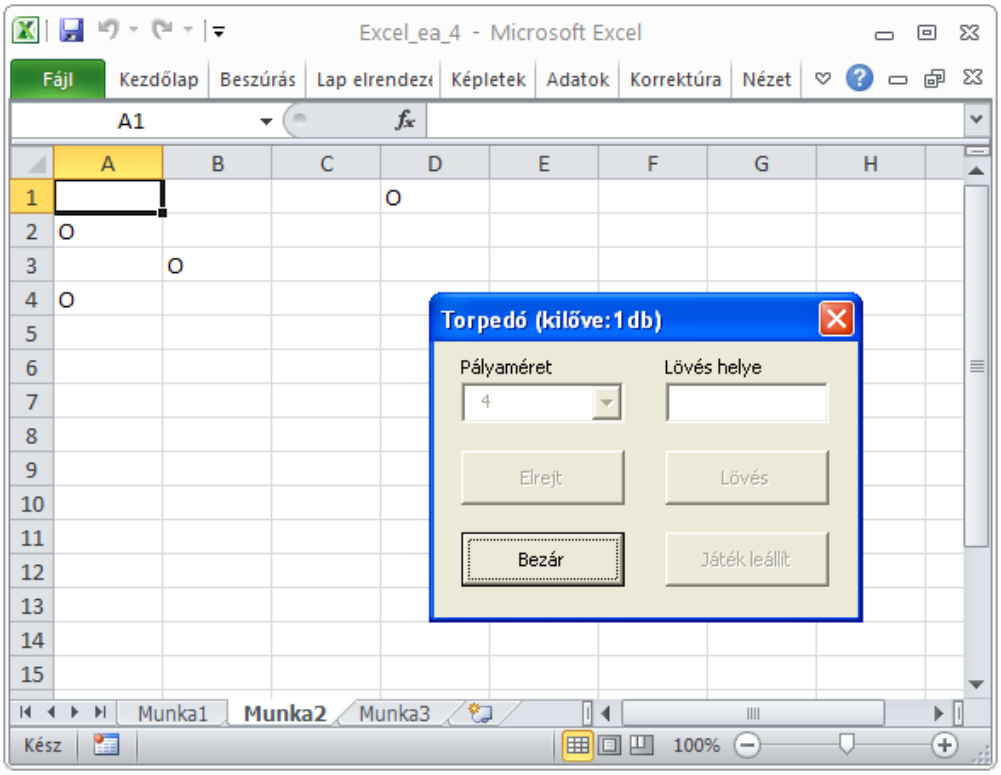


4.7. ábra. Egy már elindított játék

7. lépés: Súgó megvalósítása.

A súgó amellett, hogy aktivizálja a megfelelő munkalapot, a megfelelő vezérlők választhatóságát is beállítja. Hasonlóan a játék indítás, leállítás funkcióhoz (lásd 6. lépés), itt is a nyomógomb aktuális feliratából tudjuk meg, hogy éppen megjeleníteni, vagy bezárni kell-e a súgót.

```
'A súgó gomb tevékenységei
Private Sub btnSugo_Click()
If btnSugo.Caption = "Súgó" Then
  Worksheets(2).Activate
  btnSugo.Caption = "Bezár"
  btnLoves.Enabled = False
  tbLoves.Enabled = False
  btnJatek.Enabled = False
Else
  Worksheets(1).Activate
  btnSugo.Caption = "Súgó"
  btnLoves.Enabled = True
  tbLoves.Enabled = True
  btnJatek.Enabled = True
End If
End Sub
```



4.8. ábra. A játék súgója

8. lépés: A lövés célcellájának ellenőrzése.

A függvény a paraméterben megadott célcella címének helyességét ellenőrzi. Az adatot nagybetűsre alakítja (UCase) (így kis- és nagybetűvel is megadható a célcella oszlopának betűjele), majd ellenőrzi a hosszt, az első, illetve a második karaktert (ahol is az aktuális pályamérettel (`n`) dolgozunk). Hiba esetén a megfelelő üzenettel tájékoztatunk. A függvény eredménye a kapott célhely (`st`) helyessége (mint logikai érték).

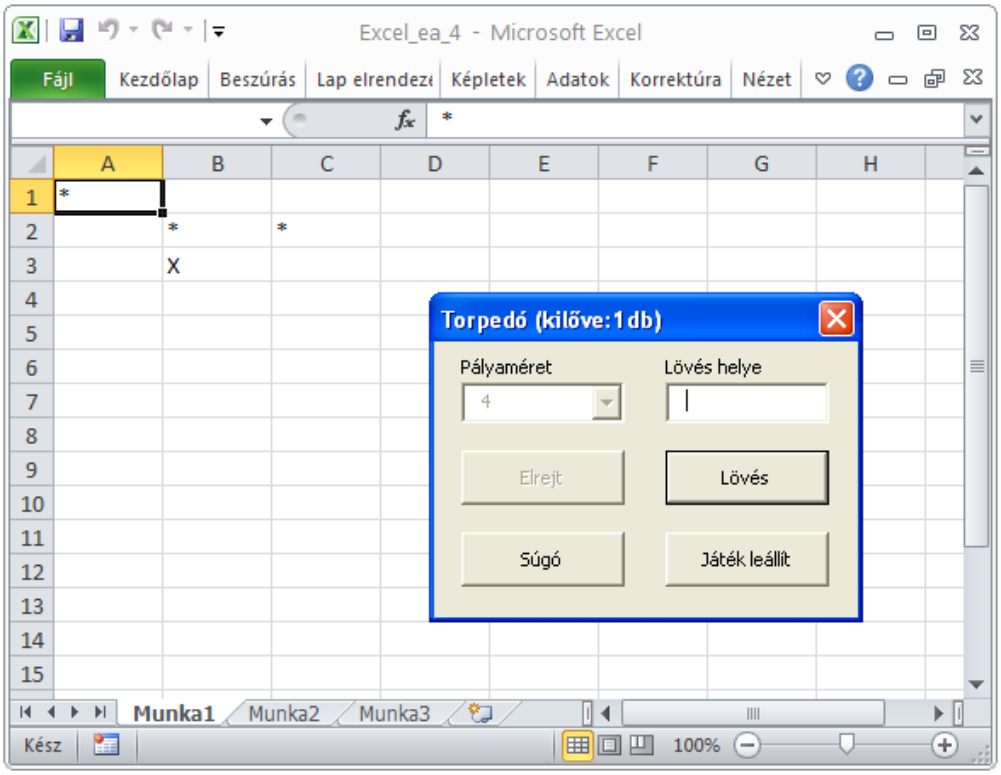
```
'A célcella helyességének ellenőrzése
Function Ellenoriz(st As String) As Boolean
Dim kar As String, ok As Boolean
ok = False
st = UCase(st)    'Nagybetűsre alakítás
If Len(st) <> 2 Then
  MsgBox ("Nem megfelelő hossz!")
Else
  kar = Chr(Asc("A") + n - 1)
  If Left(st, 1) < "A" Or Left(st, 1) > kar Then
    MsgBox ("Nem megfelelő oszlop!")
  Else
    kar = Chr(Asc("1") + n - 1)
    If Right(st, 1) < "1" Or Right(st, 1) > kar Then
      MsgBox ("Nem megfelelő sor!")
    Else
      ok = True
    End If
  End If
End If
Ellenoriz = ok
End Function
```

9. lépés: A lövés végrehajtása.

A lövés végrehajtását a beviteli mezőben megadott célhely helyességének vizsgálatával kezdjük. Ha formailag rossz a célhely, akkor az ellenőrző függvény (`Ellenoriz`) a hibáról is tájékoztat. Ha a célhely formailag helyes (azaz a pályára esik), akkor megvizsgáljuk, hogy ide lőttünk-e már korábban. Ha igen, akkor erről tájékoztatunk, egyébként meg kiértékeljük a lövést. Ha eltaláltunk egy alakzatot (a második munkalapon elrejtett alakzatok közül), akkor adminisztráljuk a találatot, frissítjük a fejlécet (ahol a kilőtt alakzatok száma megjelenik), és ha vége a játéknak (ha már kilőttük az összes alakzatot), akkor erről tájékoztatunk, és új játékot kezdünk. A találatot a `Talalt`, a mellé lövést a `Melle` adattal (lásd 2. lépés) jelezzük a célcellában.

```
'A lövés végrehajtása
Private Sub btnLoves_Click()
Dim st As String
st = tbLoves.Text
If Ellenoriz(st) Then
  'Jó a célcella címe
  If Not IsEmpty(Worksheets(1).Range(st)) Then
    MsgBox ("Ide már lőtt!")
  Else
    'Kiértékelés
    If Worksheets(2).Range(st) = Alakzat Then
      Range(st) = Talalt
      Kilott = Kilott + 1
```

```
      Call FejlecKirak
      If Kilott = n Then
        MsgBox ("A játéknak vége!")
        'A játék leállítása
        Call btnJatek_Click
      End If
    Else
      Range(st) = Melle
    End If
  End If
End If
'Lövés törlése a beviteli mezőben
tbLoves.Text = ""
'Ha még nincs vége a játéknak, akkor a beviteli mezőre állunk
if Kilott <> n Then
  tbLoves.SetFocus
End If
End Sub
```



4.9. ábra. A játék közben a pályát látjuk

Végezetül néhány ötlet (feladat) a játék továbbfejlesztési lehetőségeit illetően:

- A játéktér (pálya, háttér) kiemelése (pl. színekkel, keretezéssel), négyzetrácsos megjelenítés.
- Értékelés bevezetése (pl. számoljuk az összes lövés darabszámát, és a játék végén a talált/összes lövés arányában minősítjük a játékos teljesítményét).
- Súgó kezelése (pl. a súgót csak korlátozott számban engedjük megnézni, akkor is csak korlátozott ideig (lásd Timer függvény)).
- Hangjelzések (lásd Beep utasítás, Speech objektum, pl. Application.Speech.Speak "Great").
- Összetett, több cellából álló alakzatok kezelése (elrejtés változása, alakzat „elsüllyedése”).
- Gép elleni játék (pl. a gép is lő az általunk elrejtett alakzatokra).

5. IRODALOMJEGYZÉK

[1] Kovalcsik Géza: *Az Excel programozása.* Computerbooks, Budapest, 2005.

[2] Kuzmina Jekatyerina, Tamás Péter, Tóth Bertalan: *Programozzunk Visual Basic rendszerben!* Computerbooks, Budapest, 2006.

[3] Pusztai Pál: *Algoritmusok és adatstruktúrák,* UNIVERSITAS-GYŐR Nonprofit Kft., 2008.

[4] Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.: *Új algoritmusok.* Scolar kiadó, 2003.