2 VBA programozási ismeretek

2.1 Gondolatok a programozásról

A számítógép-program célja mindig egy felhasználó számára hasznos feladat elvégzése. A végrehajtandó feladatok, alkalmazások (*applications*) vagy egyszerűbbek, de túl sok számolni valót tartalmazva, vagy ugyanazt a rutint igen sokszor elvégezve. Érdemes programot írni akkor is, ha bonyolult, összetett számítások elvégzése és sorozatos kiíratása a feladat.

Minden feladat megoldásakor óhatatlanul felvetődik a kérdés, hogy érdemes-e erre a célra saját programot írni? Léteznek ugyanis a legkülönbözőbb célú, „testre szabható" kész programok, ezért először a piacon érdemes jól körülnézni. Egy új program kifejlesztése jóval drágább lehet, mint egy „testre szabható" program megvásárlása, de ha ezzel a megoldással nem lehet a „felhasználó" által elvárt követelményeket maradék nélkül megvalósítani, akkor egyedi program mellett kell dönteni.

2.1.1 Miért pont a Visual Basic (VBA) programot választottuk oktatásra?

Az 1990-es években sokan már temették a **BASIC** programnyelvet, mert a **C/C++** és a **PASCAL** nyelvek átvették a piac nagy részét. A **Microsoft** ekkor kiadta a **VISUAL BASIC**-et, ami a **QuickBASIC** továbbfejlesztett változataként funkcionált. Eseményvezérelt nyelvjárással és az objektumorientált programozással továbbra is fenn tudott maradni ezen a kemény piacon, hasznos és egyszerű fejlesztőeszközzé vált **Windows** környezetben. Továbbsegítette a nyelv terjedését két variánsa:

- **Visual Basic for Applications (VBA)** az **Office** programcsomag makró nyelvévé, míg a
- **Visual Basic Script** a **Windows** operációs rendszer scriptnyelvévé vált.

2002-ben újabb ráncfelvarráson esett át, és megjelent a napjaink legnépszerűbb keretrendszere a **NET**. A **Visual Basic.NET** már teljes mértékben objektumorientált volt, és csak a nevében maradt meg a BASIC szó. Szinte semmiben sem hasonlított az eredeti **BASIC program**hoz, de a fejlődés csak ilyen áron volt elérhető.

A régen sokak által lenézett, de végtelen egyszerűsége miatt a TV-komputerek révén elterjedt és megkedvelt BASIC programnyelv, a Microsoft tudatos stratégiája eredményeként - megtartva elődjének egyszerűségét és áttekinthetőségét - professzionális feladatok megoldására képes fejlesztőrendszerré vált. Készítői, egy sallangmentes programozást lehetővé tevő fejlesztői környezetet alkottak.

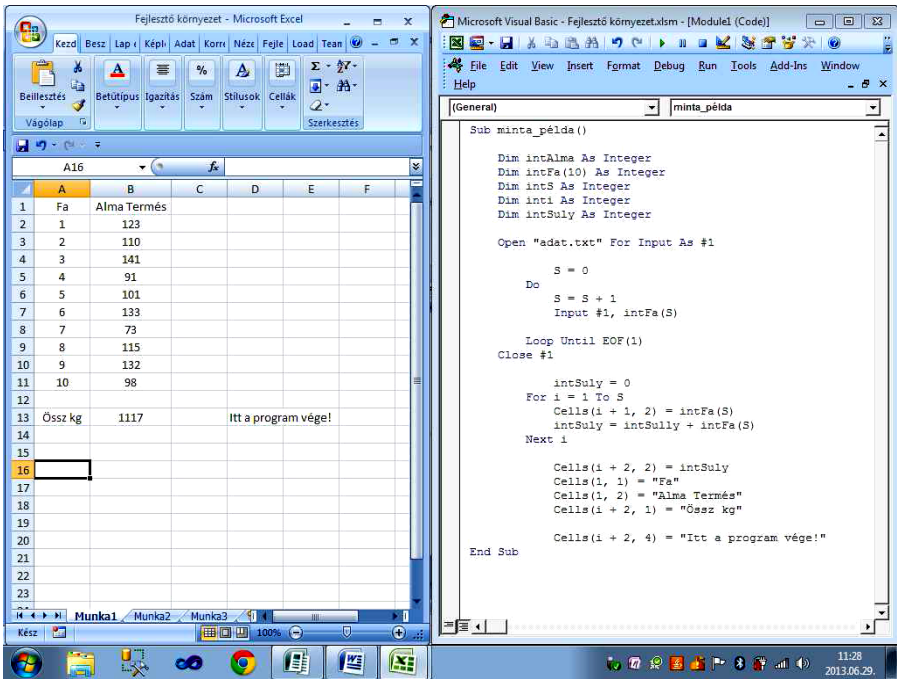
A VBA tehát minden WINDOWS **Office** programcsomag makró nyelve, így minden hallgatónak egyöntetűen rendelkezésére áll az egyetemi oktatás alatt és az otthoni számítógépen is.

2.2 VBA fejlesztő környezete

Abból a célból, hogy a programfejlesztő felhasználónak csak magára a megoldandó problémára kelljen koncentrálni, a programgyártó cégek egyre több szolgáltatást nyújtó integrált fejlesztőrendszert biztosítanak.

Az integrált fejlesztőrendszer azt jelenti, hogy egy szövegszerkesztő mögé, beépítettek több olyan programot, melyek a programíráshoz, fordításához, teszteléséhez, futtatáshoz, programmentéshez, dokumentálásához szükségesek. Ezen programok, a „szövegszerkesztő mögött” integráltan működnek együtt, a programozó minimális beavatkozása mellett.

A Windows operációs rendszer lehetővé teszi, a kétablakos programozást. A *fél szélességre állított Excel és fejlesztő ablakot egymás mellé kell helyezni.* így a Fejlesztőben írt és futtatott program eredményei, az Excel oldalon azonnal megjeleníthetők és értékelhetők.



1. ábra

Az 1. ábra bal oldali ablaka az Excel, a jobb oldali ablaka a Visual Basic fejlesztő környezet.

A jobb oldali ablakban, Modul-lapon irt program látható, mely a Sub minta_pelda() sorral kezdődik, és az End Sub sorral végződik. A programban használt azonosítók **deklarálása** a Dim utasítással kezdődő sorokban történik, majd ezt követi a program **adatfeldolgozó** része. Itt **ciklusutasításokba** rendezett értékadó parancsokat tartalmazó része található, majd Cells parancsokkal az eredmények kiíratása következik az Excel táblázatba. A programot alaposan szemlélve látható, hogy a program áttekinthetősége egyrészt szavak színes kiemelésének, másrészt a program szövegének tömbesítésével (például a „Dim” és „Cells” utasítások egymás alatt vannak, nem össze-vissza a szövegben) és a leírt szavak balról történő, különböző mértékű behúzásának köszönhető.

2.3 A VBA fejlesztőrendszer fogalmai

2.3.1 Modul lap (*Module*)

A Modul lap megjelenése egy **üres papírlap** (*űrlap*) látványát mutatja. Tudni kell azonban, hogy az itt való írást egy olyan speciális szövegszerkesztő segíti, amely nemcsak a Programírótól fogad el írásutasításokat, hanem a Fejlesztő Környezet eszközeként, a mögé integrált **Hibafelismerő** programtól is.

Természetesen a Programíró írja a programot, a Hibafelismerő csak figyeli és jelzi a programírás hibáit, majd a Programíró javításai után lerendezi és befejezi a parancssor külalakját.

A VBA program írása, e Modul lapokon történik, (lásd, a fenti ábra jobb ablaka). Tekintettel arra, hogy az egész VBA és benne a „Fejlesztő környezet és vezérlői eszközei" és így a Modul lap is angol nyelvű programozású, ezért programírás is, az angol ABC használatát, az angolszász számírást (tizedespont) követeli. A Modul lapra írt (alfa-numerikus) program leírást ezért, a VBA szabályai szerint és angol ABC-vel kell írni.

Itt újra felhívjuk a figyelmet arra, hogy az 1. ábra, bal ablakában, az **Excel, magyar** nyelvű verzióját fut, ezért itt a **magyar** írás használatos (ékezetes betűk és a tizedesjel itt vessző: „,"). Viszont a jobb ablakban **angol** nyelvű program fut, tehát a Modul lap írására is az **angol** nyelv szerinti írás kéretik (nincs ékezetes betű, a tizedesjel itt pont: „.").

2.3.2 Forrásnyelvű programleírás

A Modul lapra, a VBA szabályai szerint a sorokba írt utasítássorozat alkotja a programleírást.

Ez a felhasználó számára olvasható programleírás az **alapja,** a kiinduló **„*forrása*"**, annak a működtető programnak, melytől a tervezett számítási műveletek elvégeztetését várjuk. Ezért nevezik ezt a programírást más néven, **forrásnyelvű** programírásnak. Ezt az *alfa-numerikus* szöveget, ***forrás szö*veget** használja fel, fejlesztő környezetbe integrált automatikus **fordítóprogram** (*Compiler*). A fordító program ezt a szöveget fordítja – 2 lépésben – a számítógép számára értelmezhető **kód** sorozatokká.

Biztonsági figyelmeztetés! Itt kell felhívni a figyelmet arra, hogy ha a programjaink olyan értékesek, hogy mások számára nem akarjuk átadni, akkor ezt a **forrás** programot kell védelemben részesíteni. A VBA ezért, **jelszóval** védhető programmegnyitással tudja megakadályozni az eltulajdonítástól, vagy illetéktelen felhasználástól.

2.3.3 Fordítóprogram (*Compiler*)

A fordítóprogram feladata, hogy Modul lapon írt és készre nyilvánított **forrásprogramot**, a számítógép számára értelmezhető **hexadecimális gépi kód** sorozatokká alakítsa, és azokkal a feladatot a processzorra végrehajtassa. Ez a fordítóprogram automatikusan, a programozó közreműködése nélkül végzi a feladatát.

A VB fordítója **nem natív** (a gép által közvetlenül **nem** végrehajtható) utasításokká kódolja a forrásnyelvű szöveget, hanem egy közbenső **p-nyelvű kód**ra. A program végrehajtásakor a VB futtató magja e p-nyelvű programot interpretálja. Ennek az interpreternek feladata, hogy a p-nyelvű utasításokat **natív-kódú** utasításokká fordítsa, majd azokat a processzorral végrehajtassa. E program-végrehajtási módot, a TV-komputerek kicsi memória és tárolókapacitása kényszerítette ki, de annyira általánosan alkalmazták, hogy a p-nyelv egyfajta géptől független standarddá vált.

2.3.4 Hibafelismerő program (*Debugging*)

A VBA fejlesztőrendszer, a hibafelderítéséhez egy hatékony, ún. Hibafelismerő (*Debugging*) programot használ, mely szintén integrált része a fejlesztő környezetnek, és ez is automatikusan működik.

Itt ismét megemlékeznünk Kemény Jánosról, mert munkásságának élvezői a mai hallgatók is. „*Nemcsak azért teremtettem meg a BASIC et, hogy eggyel több számítógépes nyelv legyen. Azért csináltam, hogy a számítógép minden egyetemi hallgató (és minden diák) számára hozzáférhetővé váljék*”. Így Kemény Jánosnak tulajdonítható az automatikus Debugging használat megteremtése.

Ez az integrált hibafelismerő program valósítja meg Kemény Jánosnak azt az elhatározását: „*hogy egy olyan interaktív nyelvet fejleszt ki, amelyik rögtön reagál a használó utasítására, így lehetővé teszi, hogy minden diák vagy felnőtt, - próba-szerencse alapon - lépésről-lépésre építse föl, tapasztalja ki saját programját*”.

\- - - - - - - - -

A Hibafelismerő program azt ellenőrzi, hogy a program írása a követelményeknek megfelelő-e, valamint a tervezettnek megfelelően futatható-e, működik-e?

Az ellenőrzés **többszintű**, többfeladatú tevékenység, melyet ráadásul a programírás különböző fázisaiban kell végezni. A hibafelismerő vizsgálatokat parancssorokként, az, Enter gombbal való lezárása után automatikusan indítja. Felismert hibákról visszajelzést ad, és azoknak a programozó általi javítása után, automatikusan „lerendezi a parancssort”.

- a **szintaktikai hibák** felderítése, a nyelv szabályainak be nem tartásából eredő vizsgálat, már parancssoronként aktivizálódik;
- értelmező hibajelzéseket ad, de összetettebb nyelvi struktúrák hibafelismerésére nem mindig képes;
- a **szemantikai hibák** felderítése az algoritmus elvi hibáira, az algoritmus nem a célnak megfelelő működésére ad jelzéseket;
- a **futás közbeni hibák** felderítése is folyamatos, s ha van találat, akkor arra utaló kijelzéseket ad;
- egyes hibák olyan jellegűek, amelyek végrehajtására a program nincs felkészítve, ezért ezek a program azonnali befejezését eredményezik, ezek az ún. **run-time** (futásidejű) hibák;
- más hibák viszont ugyan nem állítják le a programfutást, csak éppen a program nem a kívánt módon működik, ezek a **program-logika** hibák.

2.4 A VBA programnyelve

2.4.1 Objektum-orientált

A Visual Basicben fejlesztett alkalmazási program: egymással kapcsolatban álló, egymásra ható objektumok rendszere. A programozás bármely része, objektuma, tárgya, eljárásokat is tartalmazó adatstruktúrája stb. összefoglalóan a program **objektuma.**

Az egyes **objektumok**nak (miként az embereknek is) vannak **tulajdonságaik** *(properties)*, - önálló **cselekvések** *(metódus)*, és a velük kapcsolatos eseményekre adott **válaszaik**, esemény-kezelő **eljárásaik** (procedures). A működő programok tevékenységeinek tárgyai mindig valamilyen objektumok.

A **tulajdonság** adja az objektum *„csomagolását”*, a **metódus** a *„magatartását”*, az **események** pedig a felhasználó és az alkalmazás közötti „*dialektikus kölcsönhatást”* írják le.

Az **objektum** informatikai értelmezése a valóság mintájára, tulajdonságokkal és viselkedésekkel felruházott egység, amit programozási szempontból tekinthetünk a hagyományos változó kibővített változatának – mintha összecsomagolva tartalmazná az érintett valóságban megjelenő tárgy számítógépes leírásához szükséges változók és alprogramok (viselkedések) összességét

Az **objektumok** lehetnek olyan elemiek, mint a gép memóriájának egy bitje, bájtja (*byte*), de lehetnek ezekből felépített, egyszerűbb vagy bonyolultabb struktúrák (űrlapok, Excel cella stb.).

A programozás során az objektumok tulajdonságai, tartalmi értékei módosíthatók, ezáltal az általuk hordozott információk igény szerint változtatható.

Az **objektum** kémiai értelmezése:

Egy kémiai feladatnál objektumnak tekinthető az anyag, mint általánosan a természetben megjelenő dolog, például folyadék. Az anyag tulajdonságokkal rendelkezik (például van térfogata, halmazállapota, színe), és viselkedik az őt érő behatások során (például megolvad). Ő maga is ki tud váltani reakciót más objektumoktól, ha változik valamelyik tulajdonsága (például ha a vizet, mint objektumot túlmelegített állapotban egy hideg üveg tárolóedénybe, mint másik objektumba öntjük, akkor kiválthatja a második objektum megrepedését.). Ha a feladat megkívánja, akkor megfogalmazhatunk további objektumokat, amelyek esetleg alkotórészei a már említett objektumoknak. Részecskék a saját objektumukra jellemző viselkedéssel és tulajdonsággal, a részecskéket alkotó további kisebb részecskék szintén egyedi viselkedéssel – itt a kisebb részek viselkedése lehet hasonló, mint az elsődleges objektum viselkedések.

2.4.1.1 Objektumok - általános elnevezési szabályai

Az objektum-orientált programozásban, minden objektumot (eljárásokat is tartalmazó adatstruktúrákat is) el kell nevezni azért, hogy azzal lehessen hivatkozni rá.

Az objektumok nevei, egyúttal **azonosítók** is, mert tulajdonképpen az azonosítók, = azonosítják, kijelölik az egyes objektumot. Tehát ezentúl amikor, az „objektumok neve” helyett „azonosítót” mondunk, akkor tudni illik, hogy az egy a nevével azonosított objektum.

Fontos ismétlés!

*A **VBA** program jelenleg csak eredeti angol nyelven hozzáférhető! Ezért programozásban létező objektumok és azonosítók (változók, speciális objektumok) elnevezésére általános szabálya az, hogy az angol ABC betűi és a matematikai írásmódja szerint kell eljárni*.

Az objektum név, egy **nagybetűvel** kezdődő, legfeljebb **255 karakter** hosszúságú szöveg, mely a második karakterétől kezdve már **számjegy**, illetve **betű** vegyesen is alkalmazható. Az írásjelek közül csak az **aláhúzás** karakter (_) lehet a nevekben. *(Ha nem kezdünk nagybetűvel, a hibaelhárító eltűri; ha számmal kezdünk vagy szóközt, más írásjelet használunk, a hibakereső kijelzi.)*

Simonyi Károly javaslata az, hogy az objektum neveknél, az úgynevezett „**Beszélő Nevek**” alkalmazása célszerű, (ne „kódszavakat” kelljen megtanulni). A **Név** lehet összetett – több szóból is álló kifejezés is, de ilyenkor az „_” aláhúzás jellel egy taggá, egy **Név**vé összefogni.

Simonyi Károly a változókra is kiterjesztette a „**Beszélő Név**” megadást még azzal, hogy a változó típusának *(lásd később)* 3 kisbetűs rövidítését, a nagybetűvel kezdődő név elé illesztette. *(pl.: intHossz).* Használata célszerű!

Tehát, **Nem** használhatók a nevek egyikében sem:

- a magyar ékezetes betűk, (Windows magyar nyelvű operációs rendszere esetén sem),
- VBA kulcsszavak (VBA parancsok, illetve VBA belső függvények nevei),
- a változók adattípusait jellemző karakterek, szóköz és pont, vagy további írásjel.

2.4.2 Strukturált programozás

A program strukturálásának alapvető célja, hogy a program fizikailag és/vagy logikailag szétbontható legyen kisebb, jól áttekinthető szegmensekre. A fizikai szétbontás komplett eljárás-könyvtárként való kezelés lehetőségét adja. A logikai szétbontás, jobb olvashatóságot segíti.

A VBA-ban a strukturálásnak 3 szintje van. A legmagasabb a **Global** szint, alatta a **Modul** szint és a legalsó szint az **Eljárás** szint. A program írásakor, az objektumok azonosítóit deklarációs utasításokkal helyezzük el a struktúra kívánt szintjeire. Az elsős hallgatók, az **Eljárás** és **Modul** szinten tanulnak programozni ebben a félévben.

2.4.2.1 Az azonosítók, láthatósága és élettartama

Az azonosítók használatát elsősorban a láthatósága és az élettartalma határozza meg. E tulajdonságokat az ún. **Deklarációs utasítás** jelöli meg. Az egyes azonosítókra a programnak csak meghatározott részein lehet hivatkozni (csak ott használhatók), e részek összességét az azonosító **hatáskörének** nevezik.

A hatáskör lehet:

- **globális,** az azonosító a program összes moduljának *(a modulokban történik a programok írása)* összes alprogramjában *(eljárásban, függvényben)* látható.
  Deklarációs kulcsszava, utasítás neve: Public;
- **modulszintű**, az azonosító csak a deklarációs utasítást tartalmazó moduljának összes eljárásában látható. Deklarációs kulcsszava, utasítás neve: Private, Dim;
- **eljárásszintű,** az azonosító kizárólag a deklarációs utasítást tartalmazó eljárásában látható.
  Deklarációs kulcsszava, utasítás neve: Dim.

Az azonosító élettartama:

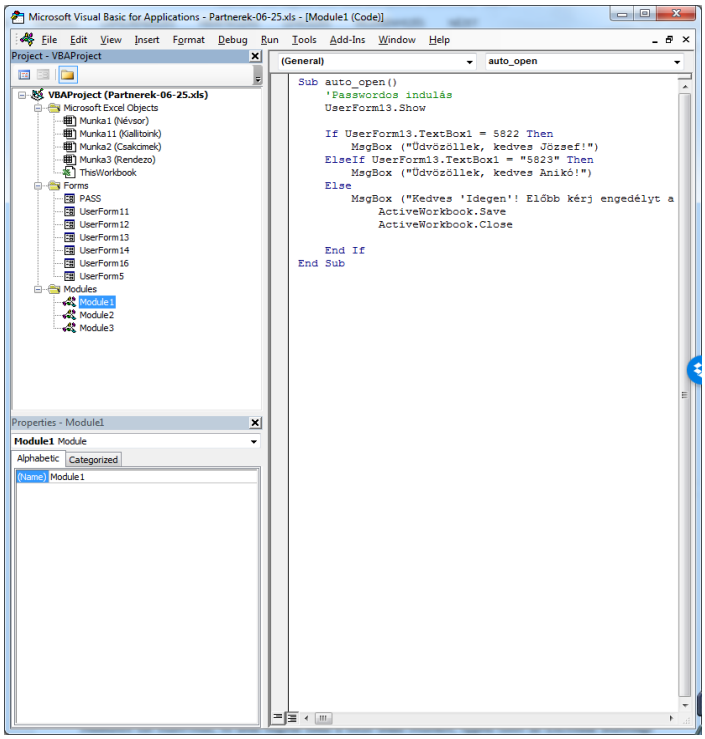
A program végrehajtásának azon periódusa, melyben az azonosító létezik, jól definiálható értékkel rendelkezik:

- **statikus**, az azonosító a program végrehajtásának teljes időszaka alatt létezik, ilyen azonosítókat deklaráló utasítás neve: Static,
- **lokális**, az azonosító csak abban az eljárásban rendelkezik jól definiált értékkel, melyben látható is. Ebből kilépve azonban az azonosító értéke definiálatlanná válik.

2.4.3 VBA – Könyvtára (Project Explorer)

Egy **Project indítása** és a fejlesztése folyamán keletkező információt, eszközt a **VBA**, egy **Project Explorer**-ben *(Program könyvtár, - Feladat Könyvtár)*, önálló fájlokban tárolja. Az Excel Munkafüzettel együtt, az **egy Project – egy Könyvtár** elvet valósítja meg, és a Windows Explorer alakú formátumban mutatja. Ezt a Könyvtárat azonban már, egy-egységként csatolja az Excel munkafüzethez. A **VBA Fejlesztő környezete** *(Jegyzet 2.2 pont)* lásd itt, a. 2. ábrán. Az Excel munkafüzetből: ***Fejlesztőeszközök*** menü / ***Visual Basic icon*** választásával indítható.

A 2. ábra bal oldalán fent látható a Project Könyvtár, alul mindig a Project Könyvtárból kiválasztott Objektum **Properties** (tulajdonság) Module ablak.



2. ábra

E minta program Projectje:

VBAProject (Partnerek-06-25.xls)

Microsoft Excel Objects

- Excel Munkafüzet lapok,

Forms

- Form objektumok (űrlapok)

Moduls

- Modul lapok.(Forrás kód írásra).

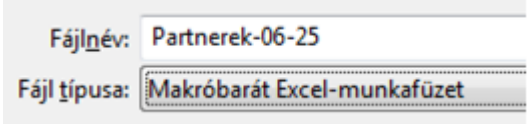
Ebből az Explorerből klikkeléssel is megnyithatók a választott objektumok. Például a Modul1-et választva, a jobb oldali szerkesztő ablakban, a Modul1 lapra írt, VBA nyelvű program.

2.4.3.1 Project Könyvtár mentése

Az Excel programok, *(a 2007-es verziótól felfelé)* a munkafüzet mentésekor, csak külön beállításra mentik el a Project Könyvtárat is, a munkafüzettel együtt.

Nagyon fontos!

Az ilyen, **Makrót** is tartalmazó Excel munkafüzet mentésekor, annak érdekében, hogy a Project Könyvtárat is mentse program, **Makróbarát-Excel munkafüzet**, vagy **Excel 97-2003** munkafüzet fájlként kell elmenteni (lásd: 3. ábra). Ebben az esetben a mentést végző program, az Excel Munkafüzethez csatolva menti el a Project Könyvtárat is úgy, hogy a Munka-könyvtárunkban egy, *.xls vagy *.xlsm kiterjesztésű fájl keletkezik.

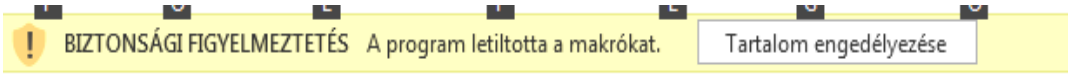


3. ábra

Mivel az Excel mentést végző programja, az általunk készített **Project könyvtárban** levő fájlokat, az Excel törzsprogramja után „**csatolva**” menti el, – ugyanúgy jár el, mintha valami **Vírus** ragadna rá, az Excel törzs végére. Az elmentett munkafüzet, *.xlsm kiterjesztést kap. Az ilyen *.xlsm kiterjesztésű fájl megnyitásakor azonban, az Excel beépített vírusvédelmi rendszere ezt észreveszi, hogy „toldalék” van a törzsprogramon. Éppen ezért az Excel **biztonsági rendszerét** kell úgy beállítani, hogy **Makró vagy Vírus** érzékelése esetén adjon értesítést a felhasználó személynek, és engedi futni a törzs utáni részeket, vagy sem. A felhasználó ilyenkor döntse el, hogy a program valós, saját Makrót tartalmaz-e, és ekkor engedélyezze a futtatást, különben Vírusirtást kezdeményezzen.

Nagyon fontos!

Az Excel megnyitása után, megjelenik a BIZTONSÁGI FIGYELMEZTETÉS (lásd: 4. ábra). A Tartalom engedélyezése a válasz.



4. ábra

A Project Könyvtárat a VBA Fejlesztő felületén lehet láthatóvá tenni. Lásd: 2.4.1. pontban leírtak.

A Project Könyvtár megjelenítése, Wiew menü / Project Explorer választással lehetséges. A **VBA** felület, bal oldalán nyílik meg, s láthatók benne könyvtárszerűen a tárolt objektumok. Excel Munkafüzet / munkalapok, Formok (ha vannak a Projectben) / és felsorolásuk, – Modul lap /és felsorolásuk.

Ebből az Explorerből is klikkeléssel megnyithatók a választott objektumok.

Hogy ne ússzunk teljesen a levegőben, most nézzük meg a VBA programnyelv programírás alapvető formai szabályait, és a tartalmi szabályait..

Eseménykezelő Eljárások: Subrutin – Function, és ezekben használatos Értékadó utasítások és Metódusok.

2.4.4 VBA program írása

A programírás során egyéni, **felhasználói eseménykezelő eljárásokat írunk**.

Ezeket az eseménykezelő eljárásokat, **Modul lapokra** írt Subrutin vagy Function rutinba kell írni. A rutinokat az Általános elnevezési szabály szerint el kell nevezni (lásd. 2.4.3.1), hogy hivatkozni lehessen rájuk. A Név után üres zárójel-párat () kell írni, vagy ebben zárójel-párba kell felsorolni, a másik rutinnak átadni szándékozott változókat. Ezt az átadást **paraméter átadásnak** hívják. A zárójelpárt akkor is ki kell tenni (üresen), ha nincs paraméter átadás. *(Ha nem tesszük ki a zárójelpárt, a Hibajavító önműködően pótolja.)*

A Modul lapokra írt:

- **Subrutint,** Sub … End Sub kulcsszavak közé kell írni (lásd 5.1 ábra).
- **Function** rutint, Function … End Function kulcsszavak közé kell írni (lásd 5.2 ábra).

2.4.4.1 A programírás formai szabályai

A Visual Basic programírás fő követelménye, az átláthatóság! A funkcionálisan azonos parancsok elkülönülése, a jobbra tartó írásmód, ezekhez az üres sorok beszúrása és Tabulátor erőteljes használata ad lehetőséget. Az alábbi irányelveket javasolt betartani (lásd 5.1 – 5.6 ábra).

```
Sub Bemutato_0()




End Sub
```
5.1 ábra

```
Function Szamol_0()




End Function
```
5.2 ábra

```
Sub Bemutato_1()
    Dim intX As Integer
    Dim sngY As Single

Elorol:
'Bemutató példa
     intX= Inputbox(„szám=?”)
        Cells(intX, 2) = intX
    Goto Elolrol

End Sub
```
5.3 ábra

```
Sub Bemutato_1()
    Dim intX As Integer

'Elágazó utasítás
if intX > 10 then
        Cells(2, 2) = intX
Else
        Cells(3, 2) = intX*2
End if

End Sub
```
5.4 ábra

```
Sub Bemutato_3()
    Dim intX As Integer
    Dim sngY As Single

       sngY = 1
    For intX = 1 To 7
       sngY = sngY + 0.5
       Cells(intX, 2) = sngY

    Next intX
End Sub
```
5.5 ábra

```
Sub Bemutato_4()
    Dim intS As Integer
    Dim sngY As Single

    s = 0
    Do
       s = s + 1
       Cells(s, 1) = s

    Loop Until intS > 7
End Sub
```
5.6 ábra

- Sub … End Sub sorába semmit nem szabad írni, kivétel a „**Címke**” és az aposztróffal kezdődő **Megjegyzés** (*Comment*) (lásd 5.3 ábra). A Comment jel (*aposztróf*) és az utána írt megjegyzéseket a program fordítója figyelmen kívül hagyja. (ide magyar ékezettel is írhattunk.)
- Minden parancsot lehetőleg külön sorba kell írni. Ha mégis több parancsot írunk ugyanabba a sorba, akkor a parancsokat kettősponttal kell elválasztani egymástól.
- Összetett parancsok kulcssorainak kezdő oszlopába pl. Elágazó parancs: If ... Else ... End If (lásd 5.4 ábra), továbbá határozott lépésű ciklus: For ... Next (lásd 5.5 ábra), valamint feltételes lépésű ciklus: Do … Loop (lásd 5.6 ábra) ne írjunk semmit, az utasításokat egy Tab-bal jobbra írjuk.

- Egymásba ágyazott, összetett parancsok írására is ugyanez vonatkozik, pl. egymásba írt két For ... Next. Ilyenkor a belső parancs egy Tab-bal már jobbra írt, majd a bennük levő értékadó utasítások újabb Tab-bal jobbra írjuk.
- Az úgynevezett szóköz (*white space*), a kód beszúrására szolgáló tabulátor és a komment jel (*aposztróf*) karakterek a kódban bárhol előfordulhatnak, kivéve a Basic kulcsszók és operátorok, valamint a programváltozó és az objektumazonosító nevek belsejét.
- Szóközöknek az utasításokon belüli írásával nem érdemes nagyon bajlódni, mert a Hibafelismerő program az utasítást a sor Enter billentyűvel történő lezárása után önműködően átformálja. Ha a sorban olyan szintaktikai hiba lenne, amit a fordító képes „ott, helyben" felismerni, azt a hibaszínnel jelzi. A helyesen írt, és felismert kulcsszavak átszíneződnek, az aposztróf és az utána álló szöveg zöld szint kap, a többi felhasználói beírás marad fekete.
- A szövegszerkesztő biztosítja még a változóneveknek az első előfordulásuknak megfelelő automatikus átformálását. is.
- Ha egy utasítást a következő sorban akarunk **folytatni**, akkor azt a sor végén jelölni kell: egy szóközt követő aláhúzás karakterrel („ _ "), ami után már semmi sem állhat.

2.4.4.2 A programírás tartalmi szabályai

Sub … End Sub kulcsszavak közé írt felhasználói eljárást tartalmilag-logikailag is fel kell építeni.

- **Deklarációs** parancssorokat a Subrutin elején kell elhelyezni. Fel kell sorolni a Subrutinban használni kívánt változókat. A deklaráció ismertetését később részletezzük.
- **Adatbevitelt** – adatfeltöltés parancsutasításokat kell a deklarációs tömb után megírni, és a változókat a programban szükséges adatokkal kell feltölteni. A deklarációban beírt váltózókat, a deklarációs metódus egyrészt létrehozza a számítógép memóriájában az igényelt memória címeken, - másrészt a létrehozott memória címeket, – szám változó esetén 0-ra törli –, szöveg esetén üres karakterekkel tölti fel. Ezért, az ide írt adatbeviteli parancssorok, az értékadó utasítások, amelyekkel az üres változókba adatokat lehet írni.
- **Adatfeldolgozás** – a programtörzs megírása követi, az Adatbevitel tömböt. E programtörzsben kell megírni, azt a felhasználói eseménykezelő eljárásunkat, mellyel a bevitt adatokat fel akarjuk dolgozni.
- **Eredmény kiírás** – tárolás, munkatömbje követi, az Adatfeldolgozás programtörzset. Az Eljárást közben keletkezett eredményeket, a Felhasználó számára meg kell jeleníteni az Excel munkalapon, s ha szükséges fájlba is el lehet menteni.

2.5 A Visual Basic utasításai és parancsai

2.5.1 Változók és konstansok deklarálása és használata

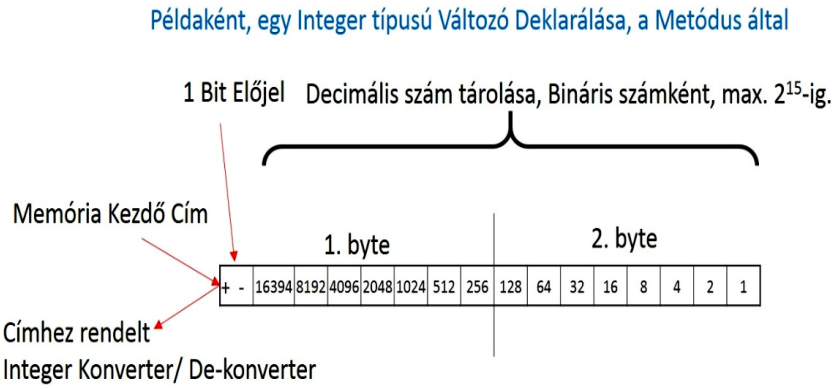
A változók és a konstansok a VBA program legegyszerűbb objektumai, mert csak egy tulajdonságuk van, és csak egy meghatározott típusú adat (szám, szöveg, logikai érték) tárolására képesek. Ezen információn kívül nincs egyéb tulajdonságuk és önálló cselekvéseik (*metódusaik*).

A változó olyan azonosító, amely az információ ideiglenes tárolására szolgál, s használatának bevezetését a változó-deklarációs utasítás adja meg.

A változók deklarációs utasításait a program, a Subrutin elején célszerű felsorolni azért, hogy a deklarációk után következő adatbevitel, adatfeltöltés programsorainál, már kész tároló helyekként szolgálhassanak.

A változók deklarációs utasításának feladata: az **Általános elnevezési szabályok** által elnevezett változónak, a kulcsszavak által meghatározott élettartamú, hatáskörű és adattípusú tárlóhelyet hozzon létre a nyelv fordítóprogramja, az élettartamukra rendelt memóriaterületen.

A példában a Dim kulcsszó behív egy **Metódust**, mely a **Változó** nevéhez hozzárendel egy **Memória Kezdő Cím**et, majd az **Integer** számábrázoláshoz szükséges, egymás utáni 2 byte bitjeit 0-ba állítja.



Ezután a Kezdő Címhez hozzárendel egy olyan **Konvertert/Dekonvertert**, amely képes forgatni, az Integer típusú Változónak, a Modul lapon, vagy Excel cellába írt Decimális (10-es számrendszerű) számait, a Memória, Bináris kódolású számábrázolásra között. Természetesen a többi típusú változó is így jön létre, csak az igényelt memóriaterület más és más.

Összefoglalva, és általánosan elmondva tudni kell, hogy az Excel cellákból vagy a Modullapon írt programból származó alfanumerikus adatokkal a számítógép, csak ezeknek a 2-es számrendszerű konverziója után képes dolgozni. Ezért tehát a VBA fordító programja az adatokat először konvertálja, majd az adatfeldolgozás során keletkező adatokat is 2-es számrendszerben tárolja, a kijelölt memória területeken. Viszont az Excel cellákban való megjelenítéshez de-konverziót, visszakonvertálást kell végrehajtani, az olvasható megjelenítés érdekében. E konverziók helyes működése érdekében, tehát előre megadni az egyes adatok „tárhelyeire” vonatkozó „típus” utasításokat is.

2.5.1.1 Alapvető adattípusok (Változók – Konstansok)

Az adattípus az állandók és a változók által hordozott információ jellegét és lehetséges értéktartományát határozz a meg. Az adattípusok az információ jellege alapján **numerikus**, **szöveges**, **logikai**, **dátum-idő** stb. csoportokba sorolhatók.

Fontosabb **numerikus** adattípusok:

- Byte: 1 bájton tárol egy [0, 255] intervallumbeli 10-es számrendszerbeli egész számot;
- **Boolean** (logikai): 2 bájton tárol egy logikai értéket? a False (Hamis) a csupa 0 bitekből álló, a True (Igaz) pedig a legalább egy bitjén nem 0 értékű duplabájtnak felel meg);
- **Integer** (egész): 2 bájton tárol egy [–32 768 ≤ x ≤ 32 767] intervallumbeli 10-es számrendszerbeli egész számot;
- **Long** (egész): 4 bájton tárol egy [–2 147 483 648 ≤ x ≤ 2 147 483 647] intervallumbeli 10-es számrendszerbeli egész számot;
- **Single** 4 bájton tárol egy (*precision floating-point*; egyszeres pontosságú lebegőpontos): az IEEE 32-bites szabvány szerint tárolja az 1,401 298 E–45 ≤ |*x*| ≤ 3,402 823 E38 feltételnek eleget tevő 10-es számrendszerbeli valós számot (Kassú lebegőpontos aritmetika);
- **Double** 8 bájton tárol egy (kétszeres pontosságú lebegőpontos): az IEEE 64-bites szabványa szerint tárolja a 4,940 656 458 412 47 E–324 ≤ |*x*| ≤ 1,797 693 134 862 32 E308 feltételnek eleget tevő 10-es számrendszerbeli valós számot (még lassúbb aritmetikájú);
- **Currency** (pénz): 8 bájton, skálázott egész számként kódolva tárol egy 922 337 203 685 477,5808 ≤ x ≤ +922 337 203 685 477,5807 intervallumbeli 10-es számrendszerbeli valós számot (gyors bináris aritmetika);
- **Decimal** (decimális): 12 bájton tárol egy kb. 28 jegyű 10-es számrendszerbeli·előjeltelen egész számot, ami a [0, 1028] határon belül változtathatóan skálázható;
- **Data (dátum):** 8 bájtos lebegőpontos szám 100.01.01 és 9999.12.31 közötti dátumot, valamint 0:00:00 és 23:59:59 közötti időpontot jelenti. A 100.01.01 és 1899. dec. 30 közötti dátumokat negatív számként tárolja, majd felette levő 9999.12.31 közöttieket pozitív számként tárolja.
- **String** (szöveges): egy 10 bájtos adatfejet követően a szöveg karakterei állnak (1 bájtban – 1 karakter), a szöveg maximális hossza kb. 2^{31} karakterben van limitálva.
- **Variant**: ez az adattípus tárolhat mind számot, mind szöveget, de memóriaigénye jóval nagyobb (szám: 16 bájt, szöveg: 22 bájtos adatfejet + szöveg hossza). *(használata nem javasolt!)*

Alapvető adattípusok táblázatos formában:

adattípus neve	jel	méret	intervallum	szám-rend-szer	szám	
byte bájt		1 Byte	[0, 255] ($256 = 2^8$)	tízes	egész	
Boolean logikai		2 Byte	0 = Hamis (False) 1 = Igaz (True)			*True*: legalább egy bit = 1
Integer egész	%	2 Byte	$-32\,768 \le x \le 32\,767$ ($32\,768 = 2^{15}$)	tízes	egész	
Long hosszú	&	4 Byte	$-2\,147\,483\,648 \le x \le 2\,147\,483\,647$ ($2\,147\,483\,648 = 2^{31}$)	tízes	egész	(kódolás, stb. mint előbb);
Single egyszeres	!	32 bit	$1{,}401\,298\,E{-}45 \le \lvert x\rvert \le 3{,}402\,823\,E38$	tízes	valós	IEEE szabvány szerint lebegőpontos, lassú,
Double dupla	#	64 bit	$4{,}940\,656\,458\,412\,47\,E{-}324 \le \lvert x\rvert$ $\lvert x\rvert \le 1{,}797\,693\,134\,862\,32\,E308$	tízes	valós	IEEE szabvány szerint lebegőpontos, lassú,
Currency pénz		8 Byte	$-922\,337\,203\,685\,477{,}5808 \le x$ $x \le +922\,337\,203\,685\,477{,}5807$	tízes	valós	bináris, gyors
Decimal tizedes		12 Byte	28 számjegy [0, 1024] skálázható	tízes	egész	előjel nélküli
Sring szöveg	$	10 Byte	$2\,147\,483\,648 =$ $= 2^{31}$ karakter	–	–	számot is szövegként tárol
Variant változó		16 Byte 22 Byte	szám, szöveg	–	–	szám: 16 Byte szöveg: 22 Byte adatfej + szöveghossz

Az alapvető adattípusú változók deklarálásának szintaxisa:

```
{Public; Private; Dim} <változó-név> As <adattípus>
```

Simonyi Károly szerinti **összetett változónév** használata javasolt. A Nagybetűvel kezdődő változónév elé 3 kisbetűs a típusra jellemző rövidítést kell írni, . *(pl.: intHossz)*. Így a program írása és futtatása közben is mindig könnyű nyomon követni, hogy melyik Változóban, milyen adatok lehetségesek.

Egyelemű Változók deklarálása

A <változónév> írására az általános elnevezési szabályok érvényesek.

A <változónév> kialakítására a *Simonyi* féle „magyar stílusú elnevezés” irányadó.

Az <Adattípus> kiválasztása az előbbi alapvető adattípusok alapján történhet.

Példák:

Egy-elemű adattárolásra képes Változó deklarálására:

```
Dim intAdat_1 As Integer
Dim lngAdat_2 As Long
Dim sngSzam1 As Single
Dim dblSzam2 As Double
Dim strSzoveg As String
```

Megadás jelekkel (az adattípus jel a név utolsó karaktere):

```
Dim intAdat_1%
Dim lngAdat_2&(5)
Dim sngSzam1!(5,5)
Dim dblSzam2#(10,4)
Dim strSzoveg$(35,2)
```

A **tömbváltozók** feladata a program által azonos kezelésmódot igénylő, azonos adattípusú nagyszámú változó egy lépésben történő deklarációja úgy, hogy mind az egyedi, mint az egy tömegben történő használatának lehetősége biztosítva legyen. Ezért a tömbváltozók deklarálása kiegészül azzal, hogy a <változó-név> után irt kerek zárójelpárba „()"- meg kell adni a változótömb méretét.

Az egydimenziós változótömb esetében a <változónév>(m) zárójelpárban egy értéket kell megadni.

A kétdimenziós változótömb esetében a <változónév>(m, n) zárójelpárban két értéket kell írni.

Egyindexes (Sor-index), vagy **egydimenziós** bemutató példák:

```
Dim intAdat_1(7) As Integer
Dim lngAdat_2(5) As Long
Dim sngSzam1(5) As Single
Dim dblSzam2(10) As Double
Dim strSzoveg(35) As String
```

Kétindexes (Sor-index, Oszlop-index) vagy **kétdimenziós** bemutató példák:

```
Dim intAdat_1(7,7) As Integer
Dim lngAdat_2(5,2) As Long
Dim sngSzam1(5,5) As Single
Dim dblSzam2(10,4) As Double
Dim strSzoveg(35,2) As String
```

Egyindexes – Dinamikus tömbváltozók deklarálása

Ha a dimenzionálásnál még nem határozható meg a tömb mérete, akkor azt üres zárójel-párral kell jelezni, hogy **Dinamikus tömböt** deklarálunk. **Egydimenziós** deklarálását bemutató példák:

```
Dim intAdat_1() As Integer
Dim lngAdat_2() As Long
```

Később a program futása során, még a változó használata előtt, a dinamikusnak deklarált változók **aktuális** méretét ReDim, vagy ReDim Preserve parancsok után, határozottá kell deklarálni.

```
ReDim intAdat_1(6) As Integer
intAdat_1(6) = 22
```

A ReDim parancssor újradimenzionálja intAdat_1 változó méretét úgy, hogy a tömb minden elemének értékét törli, majd a tömb **6.** helyére történik a „**22**" tárolása.

```
ReDim Preserve lngAdat_2(10) As Long
```

A ReDim Preserve parancssor újradimenzionálja intAdat_1(10) változót úgy, hogy a 10. memória előtti elemeket nem nullázza, s a tárolás a 10. elemre fog történni.

Konstans azonosítók

A konstans (*állandó*) olyan azonosító, amely az információ ideiglenes tárolására szolgál, s használatának bevezetését a változó-deklarációs utasítás adja meg. Értéke a program során **NEM** változtatható.

Konstans adattípusú azonosító deklarálásának szintaxisa:

{Const} <konstans-név> As <adat-tipus> = <érték>

Példák: Const intAdat_1 As Integer = 5: Const szoveg1 As String = „van”

Természetesen az adattípusok jelei a változók deklarálásánál is használhatóak.

2.5.1.2 Felhasználói adattípusok – struktúrák

A Visual Basic legpraktikusabb képességei közé tartozik, hogy lehetőség van a felhasználó által definiált adattípusok (*user defined type*) létrehozására. A felhasználói típus (melyet gyakran **struktúrának** is neveznek) egy összetett adattípus, ami azt jelenti, akár több típus felhasználásával építjük fel.

Pontosan meghatározhatjuk, hogy milyen elemek kerüljenek a struktúrába, így olyan szerkezetűre alakíthatjuk, amilyet a programunk megkíván. Az új típus definiálásához a Type ... End Type utasítást használhatjuk. Például hozzunk létre egy Személy nevű struktúrát, ennek szintaxisa a következő:

```
Public Type Szemely
    Vez_nev As String
    Ker_nev As String
    Sz_datum As Integer
    L_cim As String
    T_szam As String
End Type
```

Ekkor még nem történik helyfoglalás a memóriában, csak az egy személyhez köthető információkat tartalmazó elemek – **mezők** – használatával, egy **Szemely** típusú változóként foglalja egy csokorba úgy, hogy ez az **Adattípus** az egész alkalmazásra nézve **Globális**.

Az így létrejött **Szemely** adattípust ugyanúgy használhatjuk a változók típusdeklarációjához, mint a fejlesztőrendszer bármely más beépített adattípusát.

Ha ilyen Felhasználói struktúrát, csak modulszintű deklarációban lehet létrehozni, úgy a Public kulcsszót használjuk. Ekkor a változó hatóköre az egész Project lesz, az összes modulból használhatjuk az ilyen változókat. Ugyanilyen jelentéssel használhatjuk a Global kulcsszót is:

Nézzünk egy példát: Az előbb létrehozott **Szemely** adattípust felhasználva deklaráljunk egy „Te” változót Szemely típusnak.

```
Sub paciens()
    Dim Te As Szemely
        Te.Vez_nev = „Hollósi”
        Te.Ker_nev = „Miklós”
        Te.Sz_datum = 1942
        Te.L_cim = „1138 Párkány u. 35”
        Te.T_szam = „30/9765486”
End Sub
```

Ekkor, a Dim deklaráció hatására itt történik a tényleges helyfoglalás a memóriában, az összetett, különböző mezőknek megfelelően. Az értékadó parancssorokban ezután a Te változó mezőire, Te változónév utáni pont és a mezőnév együttesével lehet hivatkozni, pl.

```
Te.Vez_nev = „Hollósi”
```

Tömbök használata a Felhasználói adattípusban

Az új adattípus tartalmazhat tömböket is, de ezek csak **fix méretű tömbök** lehetnek. A következő példában a korábban létrehozott Szemely adattípus, (struktúra) definícióját kiegészíthetjük egy fix méretű tömbelemmel:

```
Public Type Szemely2
    Vez_nev As String
    Ker_nev As String
    Sz_datum As Integer
    L_cim As String
    T_szam As String
    Vegzettseg(3) as String
End Type
```

Most a létrehozott Személy2 adattípust felhasználva deklarálhatunk egy Te2 tömbváltozót is, Személy2 típusnak. Itt látható, hogy a Vegzettseg(3) a (0) indexel együtt, **4** indexált memóriát jelent.

```
Sub paciens2()
    Dim Te2 As Szemely2
    Te2.Vez_nev = „Hollósi”
    Te2.Ker_nev = „Miklós”
    Te2.Sz_datum = 1942
    Te2.L_cim = „1138 Párkány u. 35”
    Te2.T_szam = „30/9765486”
    Te2.Vegzettseg(0) = „Műszerész”
    Te2.Vegzettseg(1) = „Gépész technikus”
    Te2.Vegzettseg(2) = „Villamos mérnök”
    Te2.Vegzettseg(3) = „Informatikus”
End Sub
```

A Felhasználói adattípust is lehet tömbösítve deklarálni, például:

Dim Ti(10) As Szemely2 formában. A példában egy kettős For ciklussal a Ti() tömb elemeit, egy Excel munkalapról való beolvasással töltjük fel

```
Sub paciens3()
    Dim Ti(10) As Szemely2
    Dim i As Integer
    Dim k As Integer
    For i = 1 To 10
        Ti(i).Vez_nev = Cells(i, 1)
        Ti(i).Ker_nev = Cells(i, 2)
        Ti(i).Sz_datum = Cells(i, 3)
        Ti(i).L_cim = Cells(i, 4)
        Ti(i).T_szam = Cells(i, 5)
        For k = 0 To 3
            Ti(i).Vegzettseg(k) = Cells(i, k+6)
        Next k
    Next i
End Sub
```

2.6 Értékadó (*assignment*) utasítás

Az utasítás célja, hogy végrehajtásakor egy objektum valamely tulajdonságának, vagy a program egy változójának a tartalmát az általunk megadott értékre állíthassuk.

Az értékadó utasítás (*assignment statement*) műveleti jele (*operator*) az **értékadó operátor**, az egyenlőség jel (=), de itt nem a matematikai egyenlőséget jelent, hanem az Értékadó Utasítás olyan kitüntetett jele, mely szétválasztja az utasítás jobb és bal oldalát. Ezért **Szeparátornak** is lehet nevezni, mert ez mintegy „elszeparálja” a jobb és a bal oldalt.

- Az utasítás végrehajtása során **először** mindig az utasításnak az operátortól **jobbra** levő <érték> része számítódik ki (a benne szereplő változók és objektum-tulajdonságok aktuális értékeit használva), hogy **azután** értékül adja a **bal oldalán** álló változónak, vagy objektumnak.

Az utasítás pedig az alábbi formában írandó:

<objektum tulajdonság> vagy <változó> = <érték>

Ebben: <érték>

- a bal oldalán álló szintaktikai egység típusának megfelelő (vagy arra legalábbis átkonvertálható) kifejezés kell, hogy legyen (lényegében bármely konstansokat, változókat, vagy objektum-tulajdonságokat használó képlet).

Ha egy konkrét értékadó utasítás bal oldalán álló szintaktikai elem csak szöveg befogadására alkalmas, attól még VB nyelv szabályai szerint megengedik azt, hogy a jobb oldalon álló kifejezés numerikus értéket állítson elő, mert az <érték> a beírása előtt szöveggé konvertálódik.

2.6.1 Értékadó utasításában használt műveletek és függvények

2.6.1.1 Aritmetikai műveletek:

- összeadás, jele: +,
- kivonás, jele: -,
- szorzás, jele: *,
- osztás, jele: /,
- egészszám-osztás: a nem egész számok előzőleg egészre kerekítődnek, az osztás eredményének tört része elvész, jele: \,
- maradékképzés: a nem egész számok előzőleg egészre kerekítődnek, az osztás eredményének egész része elvész, parancsa: Mod,
- hatványozás, jele: ^, a ^ előtti szám a hatvány-alap, a ^ utáni szám a kitevő.

Aritmetikai műveletek táblázatos formában.

művelet	jel	példa		
összeadás	+	3 + 4	= 7	
kivonás	–	5 – 2	= 3	
szorzás	*	5 * 3	= 3·5 =15	
osztás	/	9 / 4	= 2,25	
egész osztás	\	12.51 \ 3.51	= 3	(1)
maradék	Mod()	Mod (15,4)	= 15 mod 4 ≡ 3	(2)
hatványozás	^	2 ^ 3	= 2^3 = 8	(3)
négyzetgyökvonás	Sqr()	Sqr (2)	= 1,41421	(4)

(1) az egé**szre** kerekített számokat osztja, majd az osztás eredményének is csak az egészre kerekített értékét adja.

(2) az egészre kerekített számokat egész számig osztja, egészszámú maradékát adja.

(3) a ^ (kalap) előtti szám a hatvány-alap, az utáni szám a kitevő.

(4) az Sqr kulcsszó utáni zárójelbe a négyzetgyökvonás argumentumát, pl. (2) kell beírni.

2.6.1.2 Relációs műveletek

- egyenlőség, jele: =,
- egyenlőtlenség, jele: <>,
- kisebb, jele: <,
- nagyobb, jele: >,
- kisebb vagy egyenlő, jele: <=,
- nagyobb vagy egyenlő, jele: >=.

2.6.1.3 Matematikai függvények

- Abs(<szám>): a <szám> abszolút értékét adja meg,
- Sin(<radián>): a szinusz értéket számítja ki,
- Cos(<radián>): a koszinusz értéket számítja ki,
- Tan(<radián>): a tangens értéket számítja ki,
- Atn(<szám>): az arc tg értéket számítja ki, π értéke 4*atn(1) értékével egyenlő,
- Round (<szám>[,<tizedes>]) egészre, vagy adott tizedesre kerekít.
- Fix (<szám>): <szám> egész értékre kerekítve adja meg: Fix(-4.3) értéke -4, Fix(4.3) értéke 4,
- Round (<szám>,0) parancsnak felel meg,
- Int (<szám>): <szám> egészrészét adja meg: Int(-4.3) értéke -5, Int(4.3) értéke 4,
- Exp(<szám>): az $e^{<szám>}$ (e = 2,718282) értékét adja meg,
- Log(<pozitív>): a <pozitív> szám természetes logaritmusát (ln) adja meg,

Megjegyzés: A VBA csak a természetes logaritmust ismeri, ráadásul **Log** kulcsszóval írva. Ezért a 10-es alapú logaritmust számoltatni kell. pl: $lg_{(10)}(X)$ = Log (X) / Log (10)

- Sqr(<nem-negatív>): a <nem-negatív> szám négyzetgyökét adja meg,
- Rnd: véletlen szám generálása 0 és 1 közötti értékben,

Matematikai függvények táblázatos formában.

függvény	példa	értelmezés		
Abs(<szám>)	Abs(5) Abs(–5)	= |5| = 5 = |–5| = 5		
Sin(<radián>)	Sin(1)	sin 1 rad = 0,8415		(1)
Cos(<radián>)	Cos(1)	cos 1 rad = 0,5403		(1)
Tan(<radián>)	Tan(1)	tg 1 rad = 1,5574		(1)
Atn(<szám>)	Atn(1)	arctg 1 = 0,7853 rad	$\pi = 4 \cdot \text{arctg}(1)$	(2)
Round (<szám>[,<tizedes>])	Round (3.468,1)	= 3,5		(3)
Fix (<szám>)	Fix (4.3) Fix (–4.3)	= 4 = –4	= Round(4.3,0)	(4)
Int (<szám>)	Int (4.3) Int (–4.3)	= 4 = –5		(5)
Exp (<szám>)	Exp (3) Exp (-3)	$= e^3 = 20{,}0855$ $= e^{-3} = 0{,}04795$	Exp(1) = e Exp(–1) = 1/e	(6)
Log (<pozitív>)	Log (10)	= ln 10 = 2,3026		(7)
Sqr (<nem_negatív>)	Sqr (2)	= 1,41421		(8)
Rnd ()				(9)

(1) a szöget radiánban kell megadni: x rad = $(\pi/180) \cdot y^o$ = 0.01745*y; $y^o = (180/\pi) \cdot x$ rad = 57.3*x rad

(2) a függvény egy szám arctg értékét adja radiánban.

(3) megadott tizedesre <tizedes> vagy egészre <0> kerekít.

(4) egészre csonkol.

(5) a számnál nem nagyobb, legnagyobb egész szám.

(6) $e = 2{,}7183 \ldots = 10^{-\ln 10}$ = Exp(1)

(7) $\lg x = \ln x / \ln 10 = 0{,}4343 \cdot \ln x$, valamint $\ln x = \lg x / \lg e = 2{,}3026 \cdot \lg x$

(8) nem negatív valós szám négyzetgyökét adja.

(9) véletlen szám generálása 0 és 1 intervallumban, 15 tizedes jegy pontossággal.

2.6.1.4 Konverziós függvények

- Str(<num-kifejezés>): a megfelelő decimális számot szövegesen jeleníti meg,
- Val(<szöveg>): számot tartalmazó <szöveg> szám-értékét adja meg,
- ASC(<szöveg>): <szöveg> első karakterének ANSI-kódja,
- Chr(<ANSI-kód>): <ANSI-kód> értékének megfelelő karakter,
- Hex(<egész>): az <egész> hexadecimális formáját jeleníti meg,
- Okt(<egész>): az <egész> oktális formáját jeleníti meg,
- CStr(<változó>): a <változó>-t szöveggé alakítja át,
- CSng(<változó>): a <változó>-t single típusú változóvá alakítja át (amennyiben lehetséges),
- CDbl(<változó>): a <változó>-t double típusú változóvá alakítja át (amennyiben lehetséges),
- CInt(<változó>): a <változó>-t integer típusú változóvá alakítja át (amennyiben lehetséges),
- CLng(<változó>): a <változó>-t long típusú változóvá alakítja át (amennyiben lehetséges),
- CByte(<változó>): a <változó>-t byte típusú változóvá alakítja át (amennyiben lehetséges),
- CCur(<változó>): a <változó>-t currency típusú változóvá alakítja át (amennyiben lehetséges),
- CDec(<változó>): a <változó>-t decimal típusú változóvá alakítja át (amennyiben lehetséges),
- CBool(<változó>): a <változó>-t boolean típusú változóvá alakítja át (amennyiben lehetséges),
- CVar(<változó>): a <változó>-t variant típusú változóvá alakítja át.

Konverziós függvények táblázatos formában:

Str(<num-kifejezés>)	decimális számot szövegként jeleníti meg
Val(<szöveg>)	számot tartalmazó szöveg számértékét adja meg
ASC(<szöveg>)	szöveg első karakterének **ANSI**-kódja
Chr(<ANSI-kód>)	ANSI-kód értékének megfelelő karakter
Hex(<egész>)	egészszám hexadecimális formáját jeleníti meg
Okt(<egész>)	egészszám oktális formáját jeleníti meg
CStr(<változó>)	szöveges formává alakítja
CSng(<változó>)	**Single** típusú változóvá alakítja át (ha lehetséges),
CDbl(<változó>)	**Double** típusú változóvá alakítja át (ha lehetséges),
CInt(<változó>)	**Integer** típusú változóvá alakítja át (ha lehetséges),
CLng(<változó>)	**Long** típusú változóvá alakítja át (ha lehetséges),
CByte(<változó>)	**Byte** típusú változóvá alakítja át (ha lehetséges),
CCur(<változó>)	**Currency** típusú változóvá alakítja át (ha lehetséges),
CDec(<változó>)	**Decimal** típusú változóvá alakítja át (ha lehetséges),
CBool(<változó>)	**Boolean** típusú változóvá alakítja át (ha lehetséges),
CVar(<változó>)	**Variant** típusú változóvá alakítja át.

Mintapéldák:

Kifejezés	Érték	Megjegyzés
Chr(100)	d	
Asc(„d”)	100	szám (jobbra zár)
Val(„1111, Bp. Szt. Gellért tér 4.”)	1111	(jobbra zár)
CInt(56.2)	56	(jobbra zár)
CInt(100000)	„Overflow”	hibaüzenet, túlcsordul
CStr(56.2)	56,2	szöveg
Hex(500)	1F4	
Oct(500)	764	

2.6.1.5 Logikai műveletek

- **And**: és: feltételek összefűzésére szolgál, az eredmény csak akkor igaz, ha mindkét feltétel teljesül. Például „x >= 3 And x <= 7” eredménye akkor igaz, ha „x” értéke 3 és 7 közé esik.
- **Or**: vagy: feltételek összefűzésére szolgál, az eredmény csak akkor igaz, ha bármelyik feltétel teljesül. Például „x < 3 Or x > 7” eredménye akkor igaz, ha „x” értéke 3-nál kisebb vagy 7-nél nagyobb, míg a „x >= 3 OR x <= 7” eredménye mindig igaz.
- **Xor**: kizár: feltételek összefűzésére szolgál, az eredmény csak akkor igaz, ha a feltételek között van olyan, amely teljesül és van olyan, amelyik nem. Például „x >= 3 Xor x <= 7” eredménye akkor igaz, ha „x” értéke 3-nál kisebb vagy 7-nél nagyobb.
- **Eqv**: azonos: feltételek összefűzésére szolgál, az eredmény csak akkor igaz, ha az összes feltétel igaz vagy hamis. Például az „x < 3 Eqv x > 7” akkor igaz, ha „x” értéke 3 és 7 közé esik.
- **Not**: tagad: logikai változó értékét változtatja az ellentétére, ha x = true, akkor Not x = false.

művelet	reláció	igaz, ha	például	igaz, ha
And és	összefűz	**mindkét** (egyik **is**, másik **is**) feltétel igaz	x >= 3 And x <= 7	$3 \leq x \leq 7$.
Or vagy	összefűz	**bármelyik** (egyik, másik, mindkét) feltétel igaz	x < 3 Or x > 7 x >= 3 Or x <= 7	$x < 3$ **vagy** $x > 7$ mindig
Xor kizáró vagy	összefűz	**csak** az egyik (vagy a másik, de nem mindkét) feltétel teljesül	x < 3 Xor x > 7	
Eqv azonos	összefűz	**mindkét** feltétel azonosan igaz vagy hamis	x < 3 Eqv x > 7	$3 < x < 7$
Not tagadás	előjelet fordít	eredetileg **hamis**	ha x = 1 akkor Not (x) = 0	

2.6.2 Option utasítások

Az Option szóval kezdődő utasításokat a program írására használt modul elején, az első program kezdő Sub utasítása előtt írjuk be, így az egész modulra érvényesek lesznek. Az alábbi utasítások használhatóak a programok írása során:

- **Option Explicit**: ennek az utasításnak a használata esetén csak olyan változókat használhat a program, amelyeket az Alapvető adattípusok alfejezetben leírtaknak megfelelően a programozó deklarált a változó használata előtt. E parancs nélkül a Visual Basic megengedi, hogy a változót (variant adattípussal) a program automatikusan hozza létre a változó első alkalmazásakor.
- **Option Base**: amikor a programozó Dim Tomb(3) As Integer sorral deklarálja a Tomb változót, akkor annak négy eleme lesz: Tomb(0), Tomb(1), Tomb(2), Tomb(3). Az Option Base 1, vagy Option Base 2 sor beírása esetén a Tomb változónak három, illetve két eleme lesz (Tomb(1), Tomb(2), Tomb(3); illetve Tomb(2), Tomb(3).
- **Option Compare Text**: ennek alkalmazása esetén a Visual Basic nem tesz különbséget az adott karakter „kisbetűs” és „nagybetűs” változata között. (String típusú változók vizsgálata)

2.6.3 Műveletek szöveges változóval

A String adattípus, szövegek tárolására szolgál (belsőleg Unicode kódolással). A string által foglalt memória terület egy 10 bájtos adatfejet és ezt követően a szöveg karaktereit tárolja. Szöveges változón az alábbi **öt művelet** hajtható végre:

- A **szöveghossz** vizsgálat a Len() paranccsal történik. Szintaxisa:

  <hossz_változó> = Len (<szöveg>)

A Len() parancs argumentumába (zárójelbe) beírt szöveg vagy szövegváltozó értékének megfelelő szöveg karaktereinek számát adja vissza a <hossz_változó>-ba.

- A **szöveg elejének** vizsgálata a Left() paranccsal történik. Szintaxisa:

  <szöveg_változó> = Left(<szöveg>,<karakter_szám>)

A Left() parancs argumentumába beírt szöveg vagy szövegváltozó szövegéből, a szöveg elejéről a <karakter_szám>-nak megfelelő számú karaktert ad vissza a <szöveg_változó>-ba.

- A **szöveg végének** vizsgálata a Right() paranccsal történik. Szintaxisa:

  <szöveg_változó> = Right(<szöveg>,<karakter_szám>)

A Right() parancs argumentumába - zárójelbe - beirt szöveg vagy szövegváltozó szövegéből, a szöveg végéről a <karakter_szám>-nak megfelelő számú karaktert ad vissza a <szöveg_változó>-ba.

- A **szöveg bármely részének** vizsgálata a Mid() paranccsal történik. Szintaxisa:

  <szöveg_változó> = Mid(<szöveg>,<karakter_szám>, <utána_szám>)

A Mid() parancs argumentumába beírt szövegből, a szöveg elejétől a <karakter_szám>-nak megfelelő számú karaktertől. az <utána_szám>-nak megfelelő számú karaktert ad vissza Szövegek összefűzése az & karakterrel történik. Szintaxisa:

<szöveg_változó_teljes> = <szöveg> {& <szöveg>} {& <szöveg_változó>}

A & karakter (karakterek) **összefűzik** a <szöveg>-eket, vagy <szöveg_változó>-kat egy „szövegtörzsbe”, s ez kerül át a <szöveg_változó_teljes> változóba.

név	jel	szintaxis	példa		
hossz	Len ()	<hossz> = Len (<szöveg>)	h = Len (Budapest)	*h* = 8	(1)
bal	Left ()	<bal> = Left (<szöveg>, <szám>)	b = Left (Budapest, 4)	*b* = Buda	(2)
jobb	Right ()	<jobb> = Right (<szöveg>, <szám>)	j = Right (Budapest, 4)	*j* = pest	(3)
közép	Mid ()	<köz> = Mid (<szöveg>, <szám>, <szám>)	k = Mid (Budapest, 3, 2)	*k* = da	(4)
össze-fűz	&	<szöveg> & <szöveg> &<szöveg>	of = „Bu”&”da”&”pest”		

(1) a szöveg karaktereinek darabszámát adja

(2) a szöveg számmal adott bal oldali (kezdő) karaktereit adja

(3) a szöveg számmal adott jobb oldali (végső) karaktereit adja

(4) a szöveg számmal adott balról jobb oldali (végső) karaktereit adja

Feladat: a Program olvassa be a Vegyészmérnöki Kar szöveget, és a megfelelő utasításokkal elemezze és írja ki az alábbi Excel táblában látható szórészleteket és szóösszetételt. Adja meg a szöveg hosszát, az eleje- közepe-vége 3 karakterét, és a Vegyész Kar kifejezést.

```
Sub szöveg()
        Cells(2, 1) = "Név:"
    N = "Vegyészmérnöki Kar"
        Cells(2, 2) = N
        Cells(3, 1) = "hossz:"
    h = Len(N)
        Cells(3, 2) = h
        Cells(4, 1) = "bal:"
    B = Left(N, 4)
        Cells(4, 2) = B
        Cells(5, 1) = "jobb:"
    J = Right(N, 3)
        Cells(5, 2) = J
        Cells(6, 1) = "közép:"
    k = Mid(N, 5, 2)
        Cells(6, 2) = k
        Cells(7, 1) = "összefűz:"
    of = Left(N, 7) & " " & Right(N, 3)
        Cells(7, 2) = of
End Sub
```

	A	B	C	D
1				
2	Név:	Vegyészmérnöki Kar		
3	hossz:	18		
4	bal:	Vegy		
5	jobb:	Kar		
6	közép:	és		
7	összefűz:	Vegyész Kar		
8				

2.6.4 Cellaparancsok

2.6.4.1 Cella használata írásra, olvasásra

A Cella (*Cells*) a munkalap (*WorkSheet*) és a tartomány (*Range*) objektumnak egy olyan tulajdonsága, amely paramétereiben hivatkozott cellához a megfelelő – egyetlen cellát tartalmazó – *Range* típusú objektumot rendeli. Szintaxisa:

<objektum> Cells(<sor száma>,<oszlop száma>)

Itt az objektum lehet, WorkSheets(„Munka1”), WorkSheets(1) (munkalap nevével, vagy azonosító számával) vagy Range és a <sor száma>,<oszlop száma> a cellát azonosító egész számok.

Megjegyzés:

Ha a cella szintaxisból az objektum elmarad, akkor a cellautasítás mindig az éppen aktív munkalapra lesz érvényes, s ott hajtódik végre.

A Cella megadása lehet:

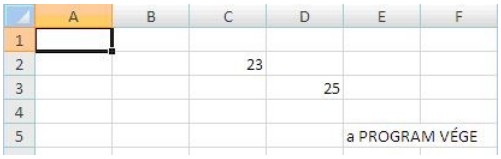
- **direkt**, a sor és oszlop számok konstans egész számok, pl. Cells (2, 5);
- **indirekt**, a sor és oszlop számok változó egész számok, pl. Cells (x, y);
- **összetett**, a sor és oszlop számok számolással képzett egész számok, pl. Cells (x + 4, y + i).

A cellautasítások, kiírások és beolvasások módját jól szemlélteti az alábbi program:

Program

```
Sub cella_muveletek()
   Dim intAdat As Integer
   Dim sngEredmeny As Single
   Dim intX As Integer
   Dim intY As Integer
      Cells(2, 3) = 23
   intX = 3
   intY = 4
      Cells(intX, intY) = 25
   intAdat = Cells(2, 3)
   sngEredmeny = Cells(3, 4) / intX
   Cells(5, 5) = „PROGRAM VÉGE”
End Sub
```

Megjegyzés



Kiírás cellába direkt címzéssel

Kiírás cellába indirekt címzéssel
Beolvasás cellából.
Beolvasás cellából művelettel.
Ajánlott a PROGRAM VÉGE megjelenítése.

2.6.4.2 Cellák tartalmának törlése

A cella és a cellatömbök tartalmának törlésének szintaxisa:

```
<objektum>.ClearContents
```

Célszerű arról gondoskodni, hogy mindig „üres” cellákba, cellatartományokba történjen a kiírás. A programfejlesztés közben nagy a valószínűsége annak, hogy egyre több „cellában felejtett kiírás” lesz látható az Excel munkalapon. Ezért gondoskodni kell arról, hogy az előző kiírás adatai le legyenek törölve a munkalapról az új adatok kiírása előtt.

- az EXCEL-ben az egérrel a mindet kijelöl gombra (bal felső sarok, az „A” oszlop fejléce mellett és az „1” sor felett), majd a billentyűzeten a „Delete” gombot lenyomva,
- VBA Programból:

```
Cells.Select
Selection.ClearContents
Range(„A1”).Select
```

Az első sor kijelöli az összes cellát, a törlésre a második parancssorban kerül sor, a harmadik parancssor csak a munkalap cellák kijelölésének megszüntetéséhez szükséges.

```
Worksheets(„Munka1”).Cells.ClearContents
Worksheets(1).Cells.ClearContents
```

Míg az előbbi esetben az Excel aktív munkalapjáról tudunk adatokat törölni, addig itt bármely munkalap tartalma törölhető.

A fenti parancsok minden cellabeírást törölnek, ezért ezekkel óvatosan kell bánni!

Az **Oszlopok és a Sorok** tartalmának törlési lehetőségeit az alábbi példák mutatják meg:

Parancssor	Megjegyzés
Columns(„A”).ClearContents	az A oszlop tartalmának törlése
Columns(1).ClearContents	az A oszlop tartalmának törlése
Rows(3).ClearContents	a 3. sor tartalmának törlése
Columns(„E:G”).ClearContents	az E, F és G oszlopok tartalmának törlése

A Worksheets(„Munka2”).Columns(1).ClearContents vagy

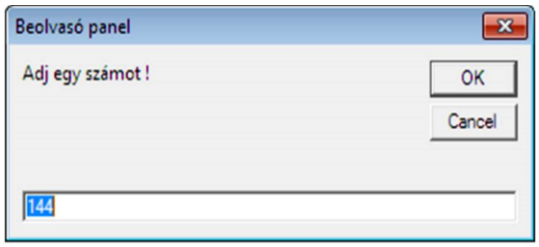
a Worksheets(5).Rows(3).ClearContents parancsokkal készített utasítások, bármely munkalap bármely oszlopának vagy sorának tartalma törölhető.

Cellák vagy **Cellaterületek** tartalmának törlése az alábbi módokon lehetséges:

Parancssor	Megjegyzés
Range(„A2”).ClearContents	A2 cella tartalmának törlése
Cells(2, 1).ClearContents	A2 cella tartalmának törlése
Cells(2, 1).Select Selection.ClearContents	A2 cella tartalmának törlése
Range(„C3:E8”).ClearContents	a C3-E8 cellatömb tartalmának törlése
Range(Cells(3, 3), Cells(8, 5)).ClearContents	a C3-E8 cellatömb tartalmának törlése

2.6.5 Adatbevitel és adatkivitel objektumokkal

2.6.5.1 InputBox - A VBA program adatbeviteli objektuma



Az InputBox (beviteli doboz) a VBA program adatbeviteli objektuma. Szintaxisa:

InputBox(Prompt, [Title], [***Default***], [XPos], [YPos], [HelpFile], [Context]) As String

- Prompt (közlés): a szürke mezőben jelenik meg, információt ad a felhasználónak arról, hogy mit várunk el tőle,
- Title (cím): a kék fejlécben jelenik meg,
- Default (induló, javasolt adat): a TextBox-ban jelenik meg, sötét alapon,

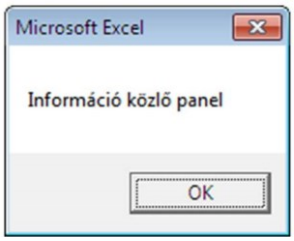
Amennyiben ezt a parancsot adjuk ki:

x = InputBox(„Adj egy számot!”,”Beolvasó panel”,144)

akkor a jobboldalon lévő ablak tűnik fel.

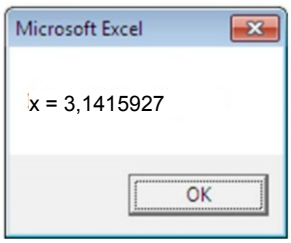
2.6.5.2 MsgBox - A VBA program adatkiviteli objektuma

MsgBox(***Prompt***, [*Buttons As* VbMsgBoxStyle = vbOKOnly], [*Title*], [*HelpFile*], [*Context*]) As VbMsgBoxResult



Ennek az objektumnak legalább egy tulajdonságát, a Prompt (közlés) tulajdonságát meg kell adni, mely a felhasználó részére szolgáltat információt és a szürke mezőben jelenik meg.

MsgBox „Információ közlő panel”



Amennyiben többféle információt szeretnénk a Promptban kiíratni, akkor az egyes részek a „&” jellel összefűzhetők. Ha például az „*x*” változó értékét szeretnénk kiírni, akkor célszerű az alábbi parancsot kiadni:

MsgBox „x = ”&x

2.6.6 Adatbevitel és adatkivitel *.txt fájl segítségével

A **text** típusú fájlokból (általában *.txt fájlok) lehet adatokat beolvasni, illetve ilyen fájlokba adatokat kiírni. A *.txt fájlok adatállománya az **ASCII** kódtábla kódjait alkalmazza. Számokon és betűkön kívül a **space**, **tab**, **enter**, **EOF** (file vége, *end of file*) vezérlő karaktereket használhatók. Open … Input ... Close parancsutasítás látja el a feladatot.

Szintaxisa:

1. sor a fájl megnyitása a művelethez:

Open <fájlnév.txt> For <alkalmazás - Input, Output vagy Append> As # <virtuális memórianév>

2. sor az alkalmazástól függően:

2.1 Beolvasás esetén:

Input # <a virtuális memória neve>, <eltárolás helye - változója>

2.2 Hozzáfűz esetén:

Append # <a virtuális memória neve>, <eltárolás helye - változója>

2.3 Kiírás esetén:

Write # <a virtuális memória neve>, <eltárolás helye - változója>

3. sor a fájl bezárása és mentése

Close # <a virtuális memória neve>

1. sor megnyitja a <fájlnév.txt>-t, vagy beolvasásra (Input), vagy kiírásra (Output), vagy hozzáfűzésre (Append), és a vezérlését átadja (As) a megnevezett virtuális memória résznek.

2.1 sor Input# parancs a kijelölt *.txt fájlból, balról jobbra haladva **karakterláncokat** olvas be a megadott változóba. Egy *karakterláncnak* tekinti az utasítás azt a szöveg vagy számkarakter sorozatot, melyet:

- **szám**, illetve „idézőjelek” között lévő **szöveg** esetén beolvasása esetén **vessző**, **pontosvessző**, **space**, **tab**, **enter**, **EOF** vezérlő karakter,
- „idézőjelek nélkül” lévő **szöveg** esetében **enter**, **EOF** vezérlő karakter zár le.

A virtuális-memória vezérlése mellett kiolvassa a megnyitott text fájlból, a **karakterláncot**, és azt eltárolja a kijelölt **változóba**.

2.3 sor Write # parancs a változó tartalmát **kiírja** a megnyitott fájlba. Ha a parancssor végén **vessző** vagy **pontosvessző** jel van, akkor a program a következő változó értékét a fájl ugyanazon sorába fogja kiírni, különben a következő változó kiírása új sorba történik. Ezek a kiírási lehetőségek jól alkalmazhatóak kétdimenziós tömbök esetén, mert így a tömb azonos soraiban lévő változók a fájlban is azonos sorba kerülhetnek, így a fájl áttekinthetőbbé válik.

3. sor Close# a még nyitott fájlt bezárja:

Open utasítás után legfeljebb annyi Input parancs lehet, mint amennyi **karakterláncot** tartalmaz a megnyitott fájl. Természetesen ciklusszervező utasítások beépítésével is (For … Next, Do … Loop), többszörözhető az Input parancsok száma.

Példa: a Szamok.txt fájl megnyitására:

```
Open „Szamok.txt” For Input As #1
```

A program a Szamok.txt fájlt az **alapértelmezett** könyvtárban fogja keresni, amennyiben nem találja, hibaüzenetet ad. Az a könyvtár számít alapértelmezettnek, amelybe az Excel fájl a megnyitása után utoljára került mentésre. Ha ismert, hogy a Szamok.txt fájl az Adatok könyvtárban található, akkor a fájl megnyitása az alábbiak szerint is történhet:

```
Open „c:\Adatok\Szamok.txt” For Input As #1
```

Ugyanakkor sokszor előfordulhat, hogy a felhasználónak kell megkeresnie a beolvasandó fájlt, vagy meghatároznia mentés helyét, így:

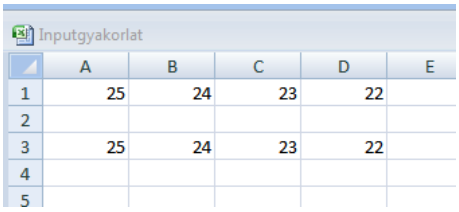
```
Open Application.GetOpenFilename For Input As #1
Open Application.GetSaveAsFilename As #1
```

Ez a Windows alapértelmezett Megnyitás és Mentés másként ablakai segítségével tehető meg.

Az Open … Close utasítás együttes működését mutatja az alábbi program. A jobb áttekinthetőség végett az összetartozó parancsok kettősponttal elválasztva ugyanazon sorban vannak.

Az Open - Close utasítás együttes működését mutatja az alábbi program. A jobb áttekinthetőség végett az összetartozó parancsok kettősponttal elválasztva ugyanazon sorban vannak.

```
Sub beolvas()
   Dim intA(5) As Integer
   Dim intB(5) As Integer
   Range(„A1:D3”).ClearContents
   Open Application.GetOpenFilename For Input As #1
      Input #1, intA(1): Cells(1, 1) = intA(1)
      Input #1, intA(2): Cells(1, 2) = intA(2)
      Input #1, intA(3): Cells(1, 3) = intA(3)
      Input #1, intA(4): Cells(1, 4) = intA(4)
   Close #1
   Open „Szamok.txt” For Input As #1
      Input #1, intB(1): Cells(3, 1) = intB(1)
      Input #1, intB(2): Cells(3, 2) = intB(2)
      Input #1, intB(3): Cells(3, 3) = intB(3)
      Input #1, intB(4): Cells(3, 4) = intB(4)
   Close #1
End Sub
```



A program az Open utasításaiban a Szamok.txt fájl adatait olvassa be, melyben csak 4 „karakterlánc” szerepel: 25, 24, 23 és 22. Az Input ezeket olvassa be sorban egy tömbváltozó elemeibe, és utána a Cells paranccsal kiírja az Excel munkalapra.

2.6.7 Elágazó utasítások. Feltételes és feltétel nélküli utasítások

Az elágazó parancsutasítások lehetővé tesznek feltétel nélkül (ún. ugró utasítás), vagy feltételtől függő működést.

2.6.7.1 GoTo utasítás

A feltétel nélküli elágazó parancsutasítás a parancssorok végrehajtásának monoton egymásutániságát képes megszakítani. Ilyen parancs a GoTo, melynek szintaxisa:

```
GoTo <címke>:
```

A <címke> egy önálló parancssor Neve, melyet kettőspont követ. E sorba más nem írható

2.6.7.2 If … Then … Else elágazó parancs

A feltételtől függően elágazó parancsutasítások a program utasításainak az írásuk sorrendjétől eltérő végrehajtását eredményezik oly módon, hogy egy logikai kifejezés aktuális értékétől függően az utasításokban leírt algoritmusban elágazásokat tesznek lehetővé. Jó példája az If … Then … Else utasítás, melynek szintaxisa:

```
If <feltétel-1> Then
   <feltétel-1 igaz utasítások>
   ElseIf <feltétel-2> Then
      <feltétel-i igaz utasítások>
   Else
      <feltétel-utolsó hamis utasítások>
End If
```

Ha a <feltétel-1> = True, akkor a Then kulcsszót követő utasítások hajtódnak végre, ellenkező esetben a program <feltétel-2> teljesülését vizsgálja. Ha <feltétel-2> sem teljesül, akkor a program továbblép, egészen addig, míg egy feltétel nem teljesül, vagy el nem jut a <feltétel-utolsó hamis utasítások> ágig.

A feltételtől függően elágazó parancsutasításnak nem kötelező „Else” ágat tartalmaznia, ekkor a program nem tesz semmit, ha egyik feltétel sem teljesül.

Mintapélda:

A Feladat az, hogy a program kérjen be egész-számokat 0 és 10 közötti értékeket, **InputBox**-szal.

A beolvasást követő adatfeldolgozásban értékelje a kapott számokat:

- ha a szám $x = 0$, akkor nem kell több számot kérni, hanem írjon értesítést a cellába és lépjen ki;
- ha a szám $x > 10$, akkor írjon értesítést a cellába és kérjen új számot;
- ha a szám $0< x < 10$, akkor írjon értesítést a cellába – abba a sorba, amekkora a szám, de értékelje azt is, hogy a szám $x > 7$, illetve $x > 3$, vagy $x < 3$. és lépjen ki.

Ezután kérjen új beolvasást!

Program

```
Sub felteteles_elagazo()
    Dim intSzam As Integer
    ujszam:
        intSzam = InputBox(„intSzam = ? (0-10)”)

    If intSzam = 0 Then
        Cells(11, 1) = „0 bevitel, nincs több beolvasás”
        GoTo vege
    End If
    If intSzam > 10 Then
        Cells(11, 1) = „10-nél nagyobb. új beolvasás”
        GoTo ujszam
    End If
    If intSzam > 7 Then
        Cells(intSzam, 1) = „Szám > 7”
    ElseIf intSzam > 3 Then
        Cells(intSzam, 1) = „Szám < 7 és Szam > 3”
    Else
        Cells(intSzam, 1) = „Szám < 3 és Szam > 0”
    End If
    GoTo ujszam
    vege:
        Cells(12,12)=”VÉGE”
End Sub
```

Megjegyzés

Alsó érték ellenőrzése

Felső érték ellenőrzése

Érték szerinti kiírás

Ugró utasítás

	A	B	C
1	Szám < 3 és Szam > 0		
2	Szám < 3 és Szam > 0		
3			
4			
5	Szám < 7 és Szam > 3		
6			
7	Szám < 7 és Szam > 3		
8			
9			
10	Szám > 7		
11	0 bevitel, nincs több beolvasás		

2.6.7.3 Select Case … End Select elágazó parancs

If …. Then feltételes utasításhoz hasonló a Select … Case utasítás, melynek szintaxisa:

```
Select Case <teszt-kifejezés>
   Case <kifejezés lista-i>
      <utasítások-i>
   Case <kifejezés lista-k>
      <utasítások-k>
   Case Else
      <máskülönben utasítások>
End Select
```

A programnak nem kötelező „Case Else” ágat tartalmaznia.

A <kifejezés> bármilyen numerikus vagy szöveges kifejezés. A <kifejezés lista-i> pedig <kifejezés-lista-i>, vagy <min-kifejezés lista-i> To <max-kifejezés lista-i>, vagy Is <alapvető-reláció-jeli> <kifejezés lista-i> formák valamelyikéből összeállított, egymástól vesszővel elválasztott lista.

Az utasítás kiértékeli <teszt-kifejezés>, és értékének vagy a <kifejezés> értékével való egyezése, vagy határaival adott intervallumba való esésével, vagy az „Is” után megadott relációs kapcsolat fennállásakor a megfelelő Case <utasítások> hajtódnak végre, hogy azok befejezése után az „End Select”-et követő utasítássor folytatódjék **Mintapélda**: Ugyan az, mint az If … Then parancsnál.

Program

```
Sub felteteles_elagazo2()
   Dim intSzam As Integer
ujszam:
   intSzam = InputBox(„intSzám = ? (0-10)”)
   Select Case intSzam
   Case 0
      Cells(11, 1) = „= 0, nincs több beolvasás”
      GoTo vege
   Case Is > 10
      Cells(11, 1) = „> 10, új beolvasás”
      GoTo ujszam
   Case Is > 7
      Cells(intSzam, 1) = „Szám > 7”
   Case Is > 3
      Cells(intSzam, 1) = „Szám < 7 és Szam > 3”
   Case Else
      Cells(intSzam, 1) = „Szám < 3 és Szam > 0”
   End Select
   GoTo ujszam
vege:
End Sub
```

Megjegyzés

Alsó érték ellenőrzése

Felső érték ellenőrzése

Érték szerinti kiírás

Ugró utasítás

	A	B	C
1	Szám < 3 és Szam > 0		
2	Szám < 3 és Szam > 0		
3			
4			
5	Szám < 7 és Szam > 3		
6			
7	Szám < 7 és Szam > 3		
8			
9			
10	Szám > 7		
11	0 bevitel, nincs több beolvasás		

2.6.8 Ciklusszervező utasítások

A ciklusszervező utasítások olyan utasítások, melyek az általuk „keretbe foglalt” utasítások csomagjának (az ún. **ciklusmagnak**) ismételt végrehajtását eredményezik. A ciklus végrehajtása lehet:

- fix számú lépésből álló ciklus : For … Next ciklus
- feltételtől függő, kötetlen számú lépésből álló ciklus: Do … Loop ciklus

2.6.8.1 For … Next ciklus

Ez a ciklusutasítás a ciklusmagot ismétli egy numerikus típusú változónak a kezdeti értékétől a végső értékéig a <lépésköz> által definiált változás mellett. A szintaxisa:

```
For <ciklusváltozó> = <kezdeti érték> To <végső érték> Step <lépésköz>
   <utasítások>
   <utasítások>
Next <ciklusváltozó>
```

A <kezdeti érték>, a <végső érték>, valamint a <lépésköz> lehet pozitív/negatív konstans vagy változó, egész vagy tört kifejezés is. Pozitív <lépésköz> esetén a <végső érték> a <kezdeti érték>-nél nagyobbnak, míg értelemszerűen negatív <lépésköz> esetén a kisebbnek kell lenni. Ha a <lépésköz> = 1, akkor elhagyható a Step.

Mintapéldák a For … Next ciklus alkalmazására:

Program

```
Sub for_next_1()
   Dim intX As Integer, intY%, intZ%
   intZ = 0
   For intX = 1 To 7 Step 0.5
      intZ = intZ + 1
      Cells(intZ + 1, 1) = intX
   Next intX
   For intY = 2 To 14
      Cells(intY, 3) = intY
   Next intY
End Sub
```

Megjegyzés

Egyszerű For … Next ciklusok

A2:A14 cellák kitöltése.

C2:C14 cellák kitöltése.

	A	B	C
1			
2	1		2
3	1,5		3
4	2		4
5	2,5		5
6	3		6
7	3,5		7
8	4		8
9	4,5		9
10	5		10
11	5,5		11
12	6		12
13	6,5		13
14	7		14

```
Sub for_next_2()
   Dim intX As Integer intY%, intZ%
   For intX = 2 To 6
      For intY = 2 To 14
         intZ = intX + intY
         Cells(intY, intX) = intZ
      Next intY
   Next intX
End Sub
```

Egymásba ágyazott For … Next ciklusok

	A	B	C	D	E	F
1						
2		4	5	6	7	8
3		5	6	7	8	9
4		6	7	8	9	10
5		7	8	9	10	11
6		8	9	10	11	12
7		9	10	11	12	13
8		10	11	12	13	14
9		11	12	13	14	15
10		12	13	14	15	16
11		13	14	15	16	17
12		14	15	16	17	18
13		15	16	17	18	19
14		16	17	18	19	20

2.6.8.2 Do … Loop ciklus

Ez a ciklus a ciklusmagot egy feltétel teljesülésétől függően ismételteti. A variációk száma az alábbiak lehetőségek kombinációjából adódik.

A ciklusmag ismétlése történhet amíg:

- a feltétel teljesül (While, amíg),
- a feltétel igazzá nem válik (Until, mígnem).

A feltétel tesztelése lehetséges:

- a ciklusmag végrehajtását megelőzően (elől tesztelő),
- a ciklusmag végrehajtását követően (hátul tesztelő).

A Do … Loop parancs szintaxisai:

Elől tesztelő ciklus

```
Do {While | Until} <feltétel>
   <utasítások>
   <utasítások>
   <utasítások>
Loop
```

Hátul tesztelő ciklus

```
Do
   <utasítások>
   <utasítások>
   <utasítások>
Loop{While | Until} <feltétel>
```

Mintapélda:

A programok Inputbox paranccsal addig olvassák be a számokat, amíg a felhasználó által megadott szám nem nulla, majd Excel munkalapon megjelenítik a beadott számokat és négyzeteiket. Az egyik program az Until, a másik a While segítségével fogalmazza meg a feltételt.

Két Mintaprogram készült:

● Sub do_ciklus_1(). Ez az Until metódust használja a függvény-vizsgálatra, és a feladatot egymás után Hátul és Elöl- tesztelős megoldással is megoldja a feladatot.

● Sub do_ciklus_2(). Ez a While metódust használja a függvény-vizsgálatra, és a feladatot egymás után Hátul és Elől-tesztelős megoldással is megoldja a feladatot.

Megjegyzés:

A Do … Loop ciklusban nincs beépített ciklus lépésszámláló, ezért – ha ez szükséges – egy változót kell alkalmazni, (pl. intS) a számlálásra. A bevezetett változó értékét a Do kulcsszó előtt nulla értékre kell állítani, majd a Do után növelni kell eggyel.

Az Elöl-tesztelt Do … Loop- nál, a Do előtt kell egy Értékadó parancs, ez a Függvényben vizsgált Változót olyan értékre állítja, amellyel biztos, hogy először belép a ciklusba a program.

Program

```
Sub do_ciklus_1()
    Dim intX As Integer, intS%
    intS = 0
Do
    intS = intS + 1
    intX = InputBox(„x=?”, „Adatbekérés”, 11)
    Cells(intS, 1) = intX
    Cells(intS, 2) = intX ^ 2
Loop Until intX = 0

    intS = 0
    intX = 12
Do Until intX = 0
    intS = intS + 1
    intX = InputBox(„x= ?”, „Adatbekérés”, 11)
    Cells(intS, 4) = intX
    Cells(intS, 5) = intX ^ 2
Loop
End Sub
```

```
Sub do_ciklus_2()
    Dim intX As Integer, intS%
    intS = 0
Do
    intS = intS + 1
    intX = InputBox(„x= ?”, „Adatkérés”, 11)
    Cells(intS, 1) = intX
     Cells(intS, 2) = intX ^ 2
Loop While intX <> 0

    intS = 0:
    intX = 12
Do While intX <> 0
    intS = intS + 1
    intX = InputBox(„x= ?”, „Adatkérés”, 11)
    Cells(intS, 4) = intX    Cells(intS, 5) = intX ^ 2
Loop
End Sub
```

Megjegyzés

a ciklus lépésszámlálója
hátul tesztelő Until ciklus

a ciklus lépésszámlálója
x változó kezdő értéke

A	B	C	D	E
11	121		7	49
12	144		8	64
13	169		9	81
0	0		10	100
			0	0

elől tesztelő Until ciklus

hátul tesztelő While ciklus

az x változó kezdő értéke

elől tesztelő While ciklus

A	B	C	D	E
11	121		7	49
12	144		8	64
13	169		9	81
0	0		10	100
			0	0

2.6.9 Programstrukturáló utasítások

A program strukturálásának alapvető célja, hogy a program fizikailag és/vagy logikailag szétbontható és később más módon összeállítható legyen kisebb, jól áttekinthető szegmensekre. A fizikai szétbontás komplett eljárás-könyvtárként való kezelés lehetőségét adja. A logikai szétbontás elsősorban a jobb olvashatóságot és tesztelhetőséget segíti.

A VBA kétféle strukturálást ismer: **eljárást** és **függvényt**. Mindkettő képes a hívás helyén számára megadott információkat átvenni, azokon módosításokat is végezni.

E programstrukturáló utasítások tárgyalásánál észre kell venni, hogy mindig **két** programról beszélünk egyszerre.

- Mindig van egy **Hívó** Subrutin, ennek a Sub neve után **üres zárójelpár** van.
- És van egy **Fogadó** – Sub, vagy Function rutin –, ezek nevei utáni zárójel-párban, vesszővel elválasztott Változó deklarációk vannak felsorolva.

Hívó Subrutin lehet akármelyik paraméter nélküli Sub. Ebben szabályosan deklarálni kell (Dim) valamennyi használni kívánt Változót, és itt kell elintézni a szükséges Adatfeltöltéseket is. Az ezt követő Adatfeldolgozási részben lehet elhelyezni az egyes jól elhatárolódó feladatokat, másik rutinokkal való elvégeztetéséhez szükséges **Hívó** utasításokat. Ezeknek a **Hívó** utasításoknak a felépítése hasonló mindkét tulajdonságú rutin meghívásakor. A programsorban a meghívandó rutin, Sub vagy Function, neve utáni kerek zárójelben () vesszővel elválasztva, fel kell sorolni a paraméterként **Átadandó** változókat. Az ide felsorolt Változók mind deklarálva kell, hogy legyenek, és a **Hívás** sorhoz érve **Aktuális értéküknek** is kell lenni. Éppen ezért, ebben a zárójelben felsorolt Változókat, Aktuális paraméterek-nek, míg a **Fogadó** oldali deklarációban szereplő Változókat, Formális paraméterek-nek nevezik.

A program működése ezek után:

1. Hívó oldali Sub fordító programja a **Hívó** sorhoz érve, felveszi a **Fogadó** rutin nevét, azzal megkeresi a kijelölt Sub vagy Function rutint, majd annak deklarációs részében levő **Formális** paraméternek **átadja** a **Hívó** oldali **Aktuális** paramétereket.

2. A **Fogadó** oldali rutin számára így már **Aktuálisak** lesznek az addig csak **Formai** paraméterek, és ezekkel így az eljárástörzsben levő parancsutasításokban el tudja végezni a programozott műveleteket.

3. A **Fogadó** rutin az eljárástörzsben levő utasításoknak végrehajtása után, a rutin végét jelentő, End Sub vagy End Function parancssor hatására, a nála **Aktuális** paraméterekkel visszaadja a vezérlést a **Hívó** Sub, hívó parancssorához. Ezután a **Hívó** Sub-ban folytatódik a feladat végrehajtás, a visszakapott **Aktualizált** paraméterekkel.

Elméleti – Működési magyarázat

A paraméter **Átadás** metódus, valójában paraméter-**átvétel**t valósít meg, mivel az Adatcsere metódusa az **Átvételi** deklarációban van megadva. Kétféle **Átadás** létezik, az **Érték szerint** és **Cím szerint**. Ez a metódus, a **Fogadó** rutin deklarációjában van meghatározva. így a **Fogadó** oldal dönti el, hogy melyik metódussal történik az **Átvétel**.

Az **Érték** szerint metódust a ByVal kulcsszó hívja, a **Cím** szerinti metódust a ByRef.

2.6.9.1 Érték szerinti paraméter átadási mód (ByVal)

A **Fogadó** rutin deklarációs zárójelben levő **Formális** paraméterek számára a típusuknak megfelelő nagyságú memóriaterület foglalódik, majd kiértékelődnek az **Aktuális** paraméterek, hogy aztán annak értékei beíródjanak a **Formális** paraméterekbe (pontosabban az imént allokált memóriaterületekre). Ily módon az eljárástörzs utasításaiban levő **Formális** paraméter hivatkozások ténylegesen a megfelelő **Aktuális** paraméterértékek másolataira vonatkoznak. Ez azzal a lényeges következménnyel jár, hogy érték szerinti paraméter-átadás esetén a formális paraméter tartalmának az eljárásban történő módosítása semmilyen tekintetben **nem** érinti a hívásban szereplő paramétert.

Az eljárásból történő kilépéskor az (End Sub vagy az End Function), kiértékelődnek az eljárás-törzsben használt **Aktuális** paraméterek, hogy aztán értékeik visszaíródjanak a **Hívó** rutin **Aktuális** paramétereibe. Ezek után az érték szerint átadott **Formális** paraméterekhez rendelt *memóriaterületek felszabadulnak,* és a hozzárendelődés is megszűnik, az eljárásbeli formális paramétereket ezáltal definiálatlan tartalmúakká teszi.

2.6.9.2 Cím-szerinti paraméter-átadási mód esetén (ByRef)

E kulcsszó specifikálásakor (vagy egyik kulcsszót sem megadva, hiszen ez az átadás alapértelmezett módja) nem értékelődik ki a hívásbeli **Aktuális** paraméter (ez ekkor csak változó lehet), hanem „csak” annak memória-területe (**Címe**) rendelődik a **Formális** paraméterhez. E **Formális** paraméterhez még itt rendelődik hozzá a **Változó Típusának** megfelelő **Konverter rutin** is. Ez a Típusnak megfelelően képes a **Decimális / Bináris** számábrázolás közötti konverziókra. Így, egy **Formális** paraméterre történő (eljárás-törzsön belüli) hivatkozás valójában a neki megfelelő **Aktuális** paraméterhez rendelt memóriaterületre történik. Valójában ez a metódus, mintegy **kölcsön adja** a Változó memóriaterületét a fogadó **Formális** paraméternek, és ennek az a lényeges következménye, hogy a Cím-szerint átadott **Formális** paraméter módosítása a megfelelő **Aktuális** paramétert is módosítja.

Az eljárásból történő kilépéskor az (End Sub vagy az End Function), az eljárás-törzsben eddig használt **Aktuális** paraméterek elvesztik paramétereiket, és ismét csak **Formális** paraméterekké válnak, és a program folytatása visszatér a a **Hívó** Sub, hívó parancssorához.

2.6.9.3 Paraméter átadás szabályai

A hívásban szereplő **Aktuális** és deklarációban levő **Formális** paraméterek megfeleltetése a listabeli sorrendjük alapján történik, és egyenlő számban kell, hogy legyenek.

A hívásban szereplő **Aktuális** és deklarációban levő **Formális** paraméterek nevei különbözzenek egymástól. Ez nem kötelező, de célszerű a **Formális** paramétereket önállóan elnevezni azért, mert e **Fogadó** Eljárást egy Projecten belül több helyről is meg lehet hívni. A különböző hívások esetén, úgysem lehet betartani a „Név-egyeztetést". Célszerű, ún. **Beszélő** neveket adni, **-be** vagy **-ki** indexel / jelzővel kiegészíteni a paraméter neveket.

Az **Aktuális** és **Formális** paraméter lista megfelelő paramétereinek, kompatibilisnek – azonos típusúaknak – (vagy, egymásba átkonvertálhatónak) kell lenni. Ez azt jelenti, hogy például egy integer típusú aktuális paraméter átadhatja az értékét egy Integer, vagy Long típusú formális paraméternek.

Paraméterezett Szubrutin, vagy Funtion? Mikor – Mit?

Mindkét Eljárás paraméterátadáson nyugszik, és mindkettőnek az **Aktuális** és **Formális** paraméterek listái, azonos szerkezetűek, de nem azonos felépítésűek. A különbség az alkalmazásban van:

- **Paraméterezett Sub hívása esetében,** az Aktuális paraméterlistában benne kell lenniük, az eredményeket visszaváró változóknak is. Átadáskor még „üresen kerülnek átadásra", majd a fogadó program tölti fel azokat eredménnyel és úgy kerülnek vissza értékkel.

A **Hívó** oldali zárójelben () felsorolt valamennyi **Aktuális** paraméter (Változó) Átadásra kerül a **Fogadó** oldali **Formális** paraméterek aktivizálása érdekében, majd az eljárástörzs parancsainak lefutása után, valamennyi paraméter értéke visszaíródik, felülíródik a hívó **Aktuális** paraméterekbe.

Ebbe a sorban, az Eljárás hívásának Szintaxisa látható. Call kulcsszóval kezdődik.

Szintaxisa: {Call} <eljárás-név> [<aktuális paraméterek listája>]

```
Sub Hivo_pr()
      Dim intA As Integer
      Dim intB As Integer
      Dim sgnC As Single
        intA = 6: intB = 2: sngC = 0

   Call Szamolo (intA, intB, intC)
      Cells(2,4) = sgnC
End Sub
```

A Hívó_pr() nevű paraméternélküli Subrutin sorai láthatók, deklarációkkal

Sub Szamolo(…) rutin hívása látható az Aktuális paraméterek átadása
A visszakapott sngC változó kiírása.

```
Sub Szamolo(intXbe As Integer, intYbe As Integer, sgnZki as Single)
   sgnZki =  2 * intXbe / 5 * intYbe
End Sub
```

A **Fogadó** Sub Szamolo(…) látható, a zárójelben a **Formális** paraméterekkel.

Az Eljárástörzs számoló feladat látható.

Az End Sub végrehajtása a paraméterek visszaadása.

- **Paraméterezett Function hívása esetén,** az Aktuális paraméterlistában csak a függvény kiszámolásához szükséges változó szerepelnek, míg a kiszámolt függvény-érték, az – úgynevezett - Függvény-változóban érkezik vissza!

A **Hívó** oldali zárójelben felsorolt valamennyi **Aktuális** paraméter (Változo) átadásra kerül a **Fogadó** oldali **Formális** paraméterek aktivizálása érdekében. De **Function** eljárástörzs parancsainak lefutása után, a kapott paraméterekből kiszámolt „Egyetlen érték" íródik csak vissza a hívó oldali parancssorba. Ez úgy lehetséges, hogy az **Aktuális** paraméterek átadásával egy időben, a **Function** „**neve**", a deklarációs zárójel után írt Változó típus deklaráció hatására, (pl. As Double) egy teljes értékű Változóvá válik, és így képes lesz érték felvételére. A **Function** eljárástörzs futása közben kiszámolt értéket ebbe a Változóvá tett, volt Function Neve Változóba tárolja be. Az End Function hatására, csak ezen Névből lett Változó, a benne levő értékkel tér vissza a hívó oldali parancssorhoz, és ott az **Aktuális** paraméter lista zárójele előtti Névbe, mint Változóba íródik be. A hívó **Aktuális** paraméterekbe, semmi sem íródik vissza.

Function rutin hívásának Szintaxisa: <változó> = <Function-név> [<aktuális paraméterek listája>]

```
Sub Hivo_pr()
      Dim intA As Integer
      Dim intB As Integer
      Dim sngC As Single
        intA = 6: intB = 2: sngC = 0

   sngC = Szam (intA, intB)
      Cells(2,4) = sngC
End Sub
```

A **Hivó_pr()** nevű paraméternélküli Subrutin sorai láthatók, Deklarációkkal.

Sub Szam (…) rutin hívása látható az **Aktuális** paraméterek átadása,

A visszakapott **Szam** változó értékének átírása sngC változóba, és felhasználása.

```
Function Szam(intXbe As Integer, intYbe As Integer)  As Single
   Szam =  2 * intXbe / 5 * intYbe
End Function
```

A **Fogadó Function Szam(……)** látható, a zárójelben a **Formális** paraméterekkel, valamint Szam-változó és a végén Típusadási deklarációja.

Az Eljárástörzs számoló feladat látható.

Az End Function végrehajtása a paraméter visszaírása.

2.6.9.4 Eljárás (Paraméteres Szubrutin)

Olyan önálló program, amely hívása céljából saját azonosító névvel és az információcsere céljából kapcsolódási felülettel rendelkezik. Az eljárás specifikálása bővebben (lásd előbb)

Az Eljárást (Paraméteres Szubrutin)-t meghívni, egy paraméter nélküli, de teljes értékű Szubrutinból lehetséges, s tartalmaz Deklarációkat, Adatbeviteleket, Eljárástörzs programsorokat. Ennek az Eljárástörzs programnak egy sora lehet, az alábbi Call hívósor.

A Sub eljárást hívó Call „eljáráshívó" szintaxisa:

```
Call <eljárásnév> ({Aktuális paraméterek listája} <változónév>, <változónév>)
```

Külön Szubrutinként írt rutin a paraméteres Szubrutin, mely a fenti **Hívás** által küldött **Aktuális** paraméterek **Fogadása** után **Aktivizálódik**, és futtatja le a saját eljárástörzsének parancssorait.

```
Sub <eljárásnév>({Formális paraméterek}<változónév>As<adattípus>, <változónév> As <adattípus>) <eljárástörzs>
End Sub
```

Az <eljárásnév> megadásakor az általános elnevezési szabályok érvényesek. Az <eljárásnév> utáni kerek zárójelek közötti rész a **Formális** paraméterek listája. Paraméterek, mert általuk az eljárás a hívásakor a működést szabályozó információkat kapnak. **Formális**, mert tényleges változókká csak az eljárás meghívásakor válnak.

- A paraméterátadás szabályait lásd előbb.
- E paraméter átadási mód, a címszerinti paraméter-átadást lásd előbb.

A Sub eljárást hívó Call parancs működését bemutató példa:

Program

```
Sub x_az_x_ediken()
    Dim intX As Integer, intY As Long
    Cells(1, 1) = „x": Cells(1, 2) = „y"
    intX = InputBox(„x = ?")
Call Sokadik(intX, intY)
    Cells(2, 1) = intX: Cells(2, 2) = intY
    Cells(5, 1) = „x": Cells(5, 2) = „y"
End Sub
```

Megjegyzés

Call hívás parancssor

	A	B
1	x	y
2	5	3125

```
Sub Sokadik(intXbe%, intYki As Long)
    Dim intI As Integer
    intYki = 1
    For intI = 1 To intXbe
        intYki = intYki * intXbe
    Next intI
End Sub
```

A For ciklus helyett az

intYki = intXbe ^ intXbe

parancs is használható.

2.6.9.5 Függvény (Function)

Ezt a rutint meghívni egy paraméter nélküli – de teljes értékű – Szubrutinból lehetséges, ennek eljárástörzsében van a **Function** hívósor. Az eljárás specifikálását bővebben lásd eéőbb.

A Sub eljárásban levő Function „eljáráshívó” szintaxisa:

<változó> = <függvénynév> („aktuális” paraméterek listája)

Az Eljáráshoz képest azzal a két különbséggel rendelkezik, hogy a neve **értéket** kap, ezzel képes az általa előállított információt a hívás helyére visszaadni, de csak ezt az egy információs értéket adja vissza. Az **Aktuális** paramétereket nem változtatja meg.

A függvények specifikálására a függvénydeklarációs utasítás szolgál, ennek szintaxisa:

```
Function <függvénynév> (<változónév> As <adattípus>) As <adattípus>
    <függvénytörzs>
End Function
```

A <függvénynév> írására az általános elnevezési szabályok érvényesek. A <függvénynév> értéket felvevő változó is, melyre a változókra előírtak is vonatkoznak, így adattípusa is van. A <függvénynév> utáni kerek zárójelek közötti rész a „**Formális** paraméterek” listája.

Program

```
Sub function_pelda()
    Dim intA(3) As Integer, intB%(3)
    Dim intN As Integer, Sk As Single
    Dim vh1 As Single, vh2!, intI%
    For intI = 1 To 3
        intA(i) = Cells(1, i + 1)
        intB(i) = Cells(2, i + 1)
    Next intI

    intN = 3
    Sk = f(intA(), intB(), intN)
    Cells(4, 2) = Sk
    vh1 = Sqr(f (intA(), intA(), intN))
    Cells(5, 2) = vh1
    vh2 = Sqr(f (intB(), intB(), intN))
    Cells(6, 2) = vh2
End Sub
```

Megjegyzés

A	B	C	D
a	1	-2	2
b	3	0	4
	11		
	3		
	5		

Skalárszorzat számolása

„a” vektorhossz számolása

„b” vektorhossz számolása

```
Function f(x%(), y%(), z%) As Single
    Dim i As Integer
    For i = 1 To z
        f = f + (x(i) * y(i))
    Next i
End Function
```

A program a futásakor az „=” jeltől jobbra levő kifejezést hajtja végre. A fordítóprogram megállapítja, hogy ott egy „f” Function változó van, keres tehát a Subrutinon kívül egy Function rutint, melynek neve „f”. A program futásakor a zárójelben levő „**Aktuális** paraméterek” (a(), b(), n) értékei átadódnak az „f” Function **Formális** paramétereinek (x(), y(), z).

A <function-törzs> utasításainak futása alatt az „f” függvényváltozó értéket kap. Az End Function sor után a program az „f” függvényváltozó értékével visszatér a hívó Subrutin hívási sorához, ezt az értéket kapja az „=” jeltől balra levő „Sk” változó. Ezt követi „Sk” változó értékének a kiírása „B4” cellába.

Tehát, **mikor, mit kell választani**?

Ha az **Aktuális** paraméterek listája olyan, hogy az átadó paraméterek mellett egynél több visszavárt paraméter is van (pl. valamilyen többértékű feladat megoldás, vagy Adatbázis olvasás), akkor a **paraméterezett Subrutin** megvalósítása a megoldás.

Ha az **Aktuális** paraméterekből feldolgozásából egyetlen adat keletkezik, pl. egy **Függvény**:

m = a*x^2 + b*x + c (az m = értéke x helyen), akkor a **Function rutin** megvalósítása a megoldás.

- * * *