Számítástechnika BSc

(Biomérnöki és Vegyészmérnöki Szak kötelező tantárgya)

VBA Programozási ismeretek

A Visual Basic for Application programozás tantárgyi követelményeinek összefoglalója. A programozás alapjai, alaki követelményei, felépítése, parancsai, eljárásai, fogalmai.

Összeállította:

Rippel Endre, Simon András, Jágerszki Gyula

Lektorálás elötti állapotú – Bővitett változat!

Bővítési idő: 2016. április 6.

Budapesti Műszaki és Gazdaságtudományi Egyetem

Vegyészmérnöki és Biomérnöki Kar

Szervetlen és Analitikai Kémia Tanszék

2013

1 Tartalomjegyzék

a 08d verzióhoz

2 Tárgyajánló

Az elsőéves hallgatókban sokszor felvetődik a kérdés, hogy a vegyészmérnök- és biomérnök-hallgatóknak szükségük van-e a Visual Basic for Applications programozás tanulására.

A válasz egyértelműen: Igen! Ez igaz még akkor is, ha a programozás tanulása kezdőknek eleinte fáradságos, de semmiképpen sem hátrányos, sőt sok szempontból nagyon is hasznos!

A programozás (bármilyen nyelven történjen is) logikus gondolkodásra nevel! Az objektumorientált nyelv, fejlesztő rendszerével végzett programozói munka hatékonyan fejleszti a rendszerszemléletű áttekintőképességet.

A fenti két képesség, (a logikus gondolkodás és rendszerszemléletű áttekintőképesség) önállóan nem tanulható, csak az ilyen gondolkodást igénylő feladatok megoldása közben sajátítható el. E képességek szintje nagyban determinálja a mérnöki tanulmányok elsajátításának eredményességét, de az igazi mérnöki munka, e képességek magas szintű művelése nélkül nem végezhető eredményesen.

Tehát a Számítástechnika tárgy témaválasztásával két előnyt egyesít:

- az egyetemi oktatásba belépő hallgatókat rögtön hozzásegíti a kívánatos mérnöki gondolkodás felfedezéséhez és fejlesztéséhez.
- az oktatásra kijelölt objektumorientált Visual Basic programnyelv, minden Microsoft Excel szervesen beépített fejlesztő rendszere, így ez minden hallgatónak rendelkezésre áll és a hallgatók otthoni munkáját is lehetővé teszi.

E gondolatsort ajánljuk minden hallgatónknak! Még az oktatás előtt mindenki meggyőzheti magát arról, hogy a félév során fellépő „esetleges nehézségeinek” legyőzésével máris a mérnöki jövőjének gyakorlásához kap elengedhetetlen eszközt!

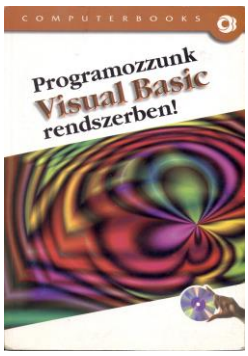
A Visual Basic fejlesztőrendszere az alapja a Microsoft Office programokba (Word, Excel, Access, PowerPoint) szervesen beépített programfejlesztő eszközöknek. Ezen összeállítás elsajátítása és gyakorlati alkalmazása a tárgy elvégzésének minimális feltétele, de nem helyettesíti a kiadott ajánlott irodalomjegyzék könyveinek tanulmányozását. Az összefoglaló csak segítséget ad, a szakkönyvek sokkal bővebb ismeretanyagának szűréséhez.

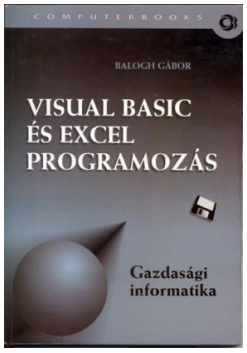
Ajánlott irodalom:

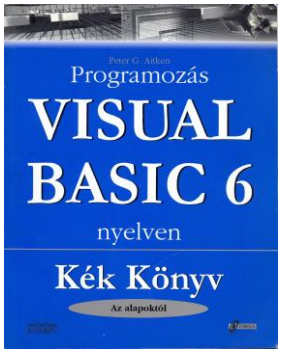
- Kovalcsik Géza: Az Excel programozása, ComputerBooks, Budapest, 2008, ISBN 963-618-332-5
- Balogh Gábor: Visual Basic és Excel programozás, ComputerBooks, Budapest, 2002, ISBN 963-618-229-9
- Kuzmina Jekatyerina, Dr. Tamás Péter, Tóth Bertalan: Programozzunk Visual Basic rendszerben, ComputerBooks, Budapest, 2006, ISBN 963-618-308-2
- Billo E. Joseph: Excel® for Chemists: A Comprehensive Guide, Wiley-VCH,New York – Chicester

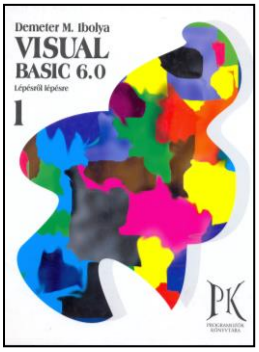
– Weinheim – Brisbane – Singapore – Toronto, 2001, ISBN 0-471-39462-9 (Paperback), 0-471-22058-2 (Electronic)

- Billo E. Joseph: Excel for scientists and engineers, John Wiley & Sons, Inc., Hoboken, New Jersey, ISBN: 978-0-471-38764-3
- Peter G Aitken: Programozás Visual Basic 6 nyelven, - Kék Könyv ISBN: 963 930 388
- Demeter M. Ibolya: VISUAL BASIC 6.0 ISMN: 963 545 218 7

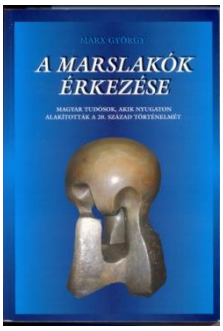








… és még egy ajánlat minden tanulmányait kezdő hallgatónknak:



Marx György: „***A MARSLAKÓK ÉRKEZÉSE***” című, 2000-ben írt könyve (Akadémia kiadó) bemutatja, hogy a 20. század robbanásszerű ipari - technikai fejlődését a Budapest iskoláiból „kirajzó” magyar tudósok milyen mértékben határozták meg.

Az informatika, a számítógép és számítástechnika, a félvezetők és a processzorgyártás, a mikrohullámú technika, az űrkutatás, a radioaktivitás, az atomkor (bombától - erőműig), a váltóáramú technika, a wolframszálas és kriptonégők, az autózástechnika, a villamos vontatás, a repüléstechnika, a biológia és az orvostudomány, a filmgyártás (Hollywood) … fejlődésének kezdete, mind - mind magyar nevekhez is kapcsolódik.

3 A számítástechnika magyar úttörői

3.1 Neumann János (Neumann János Lajos)

(Budapest, Lipótváros,1903.december 28.–Washington,1957.február 8.)matematikus.

A Fasori Evangélikus gimnáziumban tanult. 1921-ben Neumann beiratkozott a budapesti Eötvös Lóránd Tudományegyetem matematika szakára. Egyetemi évei alatt sokat tartózkodott Berlinben, ahol Fritz Habertnél kémiát, Albert Einsteinnél statisztikus mechanikát és Erhardt Schmidtnél matematikát hallgatott. Neumann 1923-ban Zürichbe ment, hogy a zürichi Szövetségi Műszaki Egyetemen vegyészetet tanuljon. Vegyészmérnöki diplomáját 1925-ben szerezte, matematikából pedig egy évvel később, 1926-ban doktorált Budapesten.1930-ban meghívták vendégprofesszornak az Egyesült Államokba, Princeton-ba. Hamarosan az ottani egyetem professzora lett (1931), majd az újonnan megnyílt a princetoni Institute for Advanced Studies professzora (1933-1955) – John von Neumann néven – ahol a világ legkiválóbb tudósai gyűltek össze. A Los Alamos-i híres 5 magyar egyike, de az elektronikus számítógépek logikai tervezésében is kiemelkedő érdemeket szerzett. Megalkotta azt a gépstruktúrát, amit ma Neumann-elvnek, Neumann típusú számítógépnek neveznek:

– a kettes számrendszer alkalmazása,

– a szerkezet logikai irányítását, azaz a műveletek megkívánt sorrendjét legelőnyösebben egy kontrol/központ [processzor] végezheti. Ha azt akarjuk, hogy a szerkezet sokoldalú legyen, akkor az éppen tárgyalt probléma speciális utasításrendszerét [software] meg kell különböztetni a kontroll/központtól [hardware], amely elvégezteti a géppel a speciális műveletet.

Tanácsadóként szerepelt az EDVAC tervezésénél. Ez volt az első olyan számítógép, amely a memóriában tárolta a programot is, 1944-től kezdték építeni, 1952-ben helyeztek üzembe.

Elméletét szabadalmaztatta, de szabad felhasználásra szétküldte az akkor ismert számítógépgyártóknak. A Neumann-elv alapján készülnek a mai számítógépek is.

3.2 Kemény János György

(Budapest, 1926. május 31. – New Hampshire, USA, 1992. december 26.), matematikus, számítástechnikus.

Kemény János**,** az 1970-es, 1980-as években az Egyesült Államokban Teller Edén kívül valószínűleg a legismertebb magyar-amerikai tudós volt**.**

Gimnáziumba a Berzsenyi Dániel Gimnáziumba járt, de 1940 januárjában a család a hitleri Németország növekvő befolyása elől külföldre emigrált. Középiskolai tanulmányait New Yorkban fejezte be, majd a Princetoni Egyetemen végzett matematikusként, és 1949-ben doktorált logikából. Előbb a Kenti Egyetem munkatársa lett.

Jellemző volt rá, hogy autójára, a „LOGIC” (LOGIKA) rendszámot íratta rá. Majd 27 évesen meghívták a Dartmouthi Főiskolára matematika-professzornak, s két év múlva a Matematikai Intézet vezetője lett. 1962-ben ő javasolta az egyetemi számítógépközpont megépítését is, a központot végül 1966-ban adták át.

Kemény János azon gondolkozott, hogyan tegye a számítógépet hozzáférhetővé egyszerre több használó számára. Amíg a használó begépel, vagy a számítógép kigépel valamit, a processzor semmit sem tesz! Ezért Kemény bevezette az időbeosztás módszerét: minden használó saját terminálján dolgozik, a központi számítógép pedig beosztja processzorának munkaidejét a használók közt. A másodperc minden törtrészét kihasználják, mindegyik használó elégedett lehet, mert úgy érezheti: a központi gép vele foglalkozik. A processzor időbeosztását nem a használók intézik, hanem maga a központi számítógép. A számítógép-időbeosztás rendszerét először a Dartmouth Kollégium vezette be (1963).

Időbeosztásos rendszeréért Kemény János kapta az IBM legelső Robinson-díját (1991).

Abban az időben a FORTRAN volt a kutatók közt legelterjedtebb nyelv, de nem volt elég emberszabású. Kemény elhatározta, hogy egy 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. Megfogalmazta elvárásait:

- A programozási nyelv legyen alkalmas eltérő célok kielégítésére.
- A nyelv kezdők számára is könnyű legyen.
- A magasabb szintű utasításokat csak később kelljen megtanulni.
- A használó és a gép között a nyelv legyen interaktív.
- Könnyen érthető hibajelzéseket adjon.
- A speciális gép-architektúra ismerete nélkül is lehessen használni.
- Óvja meg a használót a gép operációs rendszerének problémáitól.

Így született meg Kemény János és Tom Kurtz kezében 1964-ben a BASIC nyelv.

„Nem csak 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”. Az első BASIC program 1964. május 1-jén reggel 5 órakor futott.

(Kemény és Kurtz levédte a BASIC nevet, de mindenki szabadon használhatta-adaptálhatta a nyelvet. Így a BASIC a 20. század második felének legelterjedtebb programozási nyelve lett.)

Kemény János egyébként a ma közkedvelt elektronikus levelezés (e-mail) úttörője volt. Felesége egy 200 km távolságban levő főiskolán dolgozott. A két főiskola központi gépének összekapcsolásával létrejött az első „internet” amelyen keresztül levelezhettek.

3.3 John Barden

(Madison, Wisconsin, 1908. május 23. – Boston, 1991. január 30.)amerikai fizikus.

Wigner Jenő tanítványa, 1947-ben megalkotta a vákuum-elektroncsöveket felváltó tranzisztort, mely egy germánium kristály félvezető elem (mérete néhány mm^2).

Az egyetlen ember, aki kétszer kapta meg a fizikai Nobel-díjat:

- 1956-ban a tranzisztor feltalálásáért William Bradford Shockle-lal és Walter Brattainnel együtt, és
- 1972-ben a konvencionális szupravezetés elméletért Leon Neil Cooperrel és John Robert Schriefferrel együtt.

3.4 Gróf András (Andy Grove, Andrew Steven Grove)

(Budapest, 1936. szeptember 2. –), vegyészmérnök, az Intel Corporation társalapítója.

Felsőfokú tanulmányait az Eötvös Loránd Tudományegyetem kémikus szakán kezdte. 1956-ban, a forradalom leverése után az Egyesült Államokba emigrált. 1960-ban végzett vegyészmérnökként a New York City College-ban, majd a Berkeley Egyetemen szerzett doktorátust.

A PhD-fokozat megszerzését követően 1963 és 1968 között a Fairchild Semiconductor kutatás-fejlesztési részlegénél dolgozott.

1969-ben, Gordon Moore társaként megalapította az Intel Corporationt, amely ma a világ legnagyobb félvezetőgyártó vállalata. 1979-től a cég elnöke, 1987-től vezérigazgatója, 1997-től pedig a vállalat vezérigazgatója és igazgatótanácsának elnöke egy személyben. A vezérigazgatói posztról 1998-ban lemondott, de 2005 májusáig az igazgatótanács elnöke maradt. Azóta rangidős tanácsadó az Intel munkájában.

A Neumann-féle számítógépben a mikroprocesszor számol, az gondolkozik, mialatt a számítógép többi szerve tárolja, megjeleníti, akár ki is nyomtatja az információt.

Ötven évvel ezelőtt az ENIAC-ban 17000 vákuum-elektroncső dolgozott, de a tranzisztor alkalmazása forradalmasította az elektronikai lehetőségeket.

1982-ben az INTEL által gyártott „286” jelzésű mikroprocesszorban már 130000 tranzisztor dolgozott, 130000 elektroncsővel egyenértékű módon, de sokkal gyorsabban. Három évvel később 1985 októberében- a „386” jelzésű mikroprocesszorban 275000 tranzisztor volt. További 3 év múlva (1989 áprilisában) a „486” jelű mikroprocesszorban 1200000 tranzisztor számolt. Három év elteltével 1993 márciusában-jött ki a piacra az INTEL 586 mikroprocesszor, becenevén PENTIUM, ami 3100000 tranzisztor munkáját végzi el. A következő 686 processzor-lapkában 1995. november-, aminek beceneve PENTIUMPRO, már 5500000 tranzisztor dolgozik. A fejlesztés egyre gyorsult, a 20. század 10 millió tranzisztorral ekvivalens mikroprocesszorral zárt. Az INTEL vállalat (értsd: INTelligens ELektronics) kezében van a világ mikroprocesszor-piacának 75%-a, évi forgalma meghaladja a 10 milliárd dollárt.

A Kék Óriás, az IBM, - INTEL mikroprocesszort és MICROSOFT operációs rendszert használ.

3.5 Simonyi Károly (Charles Simonyi)

(Budapest, 1948. szeptember 10. –) szoftverfejlesztő, a szándékorientált programozás (Intentional Programming, IP) kutatója.

Második magyarként kétszer is járt a világűrben. Ő volt az ISS eddigi ötödik és hetedik űrturista-látogatója.

A számítástechnikával először középiskolásként került kapcsolatba, amikor éjjeliőrként egy szovjet Ural–2 típusú elektroncsöves számítógép vezérlőtermére vigyázott. Az egyik mérnök megtanította a gép programozására, és az ifjú Károly 18 éves korában már fordítóprogramokat készített, sőt egyik programját egy állami vállalat megvásárolta. 1968-ban az Amerikai Egyesült Államokba költözött.

1981 októberében, amikor a MICROSOFT-nak még csak 32 alkalmazottja volt, a cég munkatársa lett. (Ma a MICROSOFT-nak több tízezer alkalmazottja van.)

A korai 1980-as években APPLE-MICROSOFT együttműködésben, Steve Jobs, Bill Gates és Charles Simonyi munkája nyomán megszületett a barátságos MACINTOSH számítógép színes grafikával és egérrel.

Simonyi Károly első nevezetes alkotása volt a MULTIPLAN táblázatszerkesztő (spreadsheet), menüvel indítva. BRAVO-tapasztalataira támaszkodva 1981-ben megkezdte a WORD szövegszerkesztő kidolgozását egéralkalmazásával, ami már többféle betűtípust ajánl, és munka közben lehetőséget kínál a szerkesztés alatt álló szöveg tördelésének megtekintésére.

A MICROSOFT elnöke, Bill Gates kijelentette: „Meg fogjuk teremteni a világ legszebb táblázatszerkesztőjét!”, Simonyi Károly és Jabe Blumental valóban létrehívta az EXCEL-t.

Simonyi Károly vezette be a programozásba a "magyar stílusú elnevezést": az egyes adatcsomagok elnevezésére nem rövid és értelmetlen betűszavakat ajánlott, mint például blabla, nem is hosszú magyarázkodó nevet, hanem a név első része az adattípust, második része az adat jelentését adja. Peter Norton melegen ajánlja ezt a "magyar stílust".

Simonyi Károly és Scott McGregor megalkotta_a „Windows” (ablakok) operációs rendszert: "ablakokat" tárt ki, hogy bepillanthassunk a számítógép agyába. A WINDOWS-rendszer előnye, hogy csatlakoztatható a világ minden számítógéptípusára, függetlenül a gépet gyártó vállalattól. Ma a személyi számítógépek 90%-a MICROSOFT operációs rendszert használ.

Simonyi Károly a MICROSOFT fő rendszerépítésze lett, aki arra tanította az embereket, hogy a számítógép nem csak egy szám-végeredmény elérésére és szövegszerkesztésre szolgálhat, hanem a kölcsönhatásra alapozott grafikus rendszerével virtuális valóságot teremthet.

2002-től többedmagával saját vállalkozást indított, és már nem alkalmazottja a MICROSOFT-nak.

3.6 Szentiványi Tibor

(1931 – 2009) okleveles elektromérnök, találmánya a nagy floppy**.**

Szentiványi Tibor több hazai és nemzetközi szakmai szervezet tagja, - a Kiss Áron Magyar Játék Társaság, alapító elnöke, a Pro Ludo-díj alapítója és adományozója**.**

Igazi polihisztor volt, mert a játékokon kívül sokféle terület szakértője volt: informatika, pénz (kultúr- és technikatörténet), de tartott előadásokat, publikált sok más területen is.

3.7 Jánosi Marcell

(Budapest, 1931. december 5. – 2011. július) gépészmérnök, konstruktőr.

A mikro floppy feltalálója. Az MCD típusú hajlékonylemezt a Budapesti Rádiótechnikai Gyárban (BRG) fejlesztette ki. A BRG-ben részt vett az első modern, háromsebességes orsós magnó, a Calypso megtervezésében, nem is beszélve az akár winchesterként is használható magnetofonmechanikáról.

A 3,5"-es méretű hajlékonylemez rendszert - 3 hüvelykes kivitelben - Jánosi Marcell dolgozta ki a Budapesti Rádiótechnikai Gyárban 1973-ban. A lemezt és a hozzá tartozó BRG MCD-1 típusjelű meghajtót szabadalmaztatták. Azonban később a Jánosi-féle floppy maradt technikatörténeti kuriózum - a nemzetközi szabadalmat az állam nem újította meg, így az elveszett!

A 3 hüvelykes floppy-t 1982-től a japánok (többek között a Hitachi és a Matsushita) gyártották, és az Amdek nevű cég forgalmazta, elsősorban az Apple II-ben. A japánok bevallottan "Mr. Jánosi" találmányát felhasználva vitték végig a floppy ügyét.

Jánosi Marcell a magyar hardveripar Rubik Ernője volt: ugyanabban az évben, 1974-ben, amikor Rubik előállt a bűvös kockával, Jánosi feltalálta a 3 hüvelykes floppy lemezt, és ezzel örökre beírta magát a számítógépes perifériák történelmébe.

3.8 Dr. Náray Zsolt

(1927–1995) a Számítástechnikai Koordinációs Intézet igazgatója. 1985-től Számítástechnikai Kutató Intézet és Innovációs Központ – **SZKI** alapító főigazgatója.

Dr. Náray Zsolt a világsikernek számító OCR programok, az Optikai Karakterfelismerő programok, a „Recognita” és a „MProlog” rendszer megalkotója.

3.9 Dr. Kovács Győző

(1933 – 20102) magyar villamosmérnök, számítástechnikus, informatikus, az informatikai kultúra jeles terjesztője. Az első magyarországi PC-gyár, az Sci-L igazgatója. A hazai számítógépgyártás irányitója és Ő szervezte a MTV-ben az első távtanulási tanfolyamot, a TV-BASIC-et, így először lehetett távtanulási formában programozói képesítést szerezni.

4 Tantárgyi információk

Kötelező tantárgy

Tantárgyadatlap

Tantárgykövetelmények

2013. szeptember

SZÁMÍTÁSTECHNIKA

Tantárgy kódja	Szemeszter	Követelmény	Kredit	Nyelv	Tárgyfélév
BMEVESAA103	1	0+2+0 f	2	magyar	1/1

A tantárgyfelelős személy és tanszék:

Dr. Simon András, Szervetlen és Analitikai Kémia Tanszék

A tantárgy előadója:

Dr. Simon András, egyetemi adjunktus

A tantárgy célkitűzése:

Algoritmizálási készség fejlesztése egy programnyelv elsajátításán keresztül. Alapvető mérnöki számítások elvégzését, eredmények megjelenítését segítő szoftverek (pl. táblázatkezelő) készségszintű használata.

A tantárgy részletes tematikája:

EXCEL - A táblázatkezelés alapműveletei. Cellahivatkozások, cellanevek. Adattípusok, adatmozgatás, formázás. Számolás cellákkal, függvények alkalmazása. Adatok ábrázolása. Adatsorra függvény illesztése ("trendvonal").

VISUAL BASIC FOR EXCEL - A soros programozás alapjai. Változó fogalma, típusok, kifejezések, értékadás, feltételes és ciklusszerkezetek, adatbeolvasás, -kiíratás. Blokkdiagram. Szintaxis diagram. Tömb, rekord, keresés, rendezés. Műveletek egy- és kétindexes tömbökkel. Változók hatásköre, lokális és globális változók. Eljárás- és függvény deklaráció, paraméterátadás. Makrók rögzítése és átalakítása Szövegfájlok használata.

5 VBA programozási ismeretek

5.1 Gondolatok a programozásról

A számítógép-program célja mindig egy felhasználó számára hasznos feladat elvégzése. A végrehajtandó feladatok (alkalmazások, applications) lehetnek egyszerűbbek, de sok számolni valót tartalmazóak, vagy ugyanazt a rutint kell sokszor elvégezni. 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 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 mégis az egyedi program mellett kell dönteni.

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 QuckBASIC 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, 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 programhoz, 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.

Visual Basic for Applications (VBA) tehát minden WINDOWS Office programcsomag makró nyelvéve, így minden Hallgatónak egyöntetűen rendelkezésére áll az egyetemi oktatás alatt és az otthoni számítógépen is.

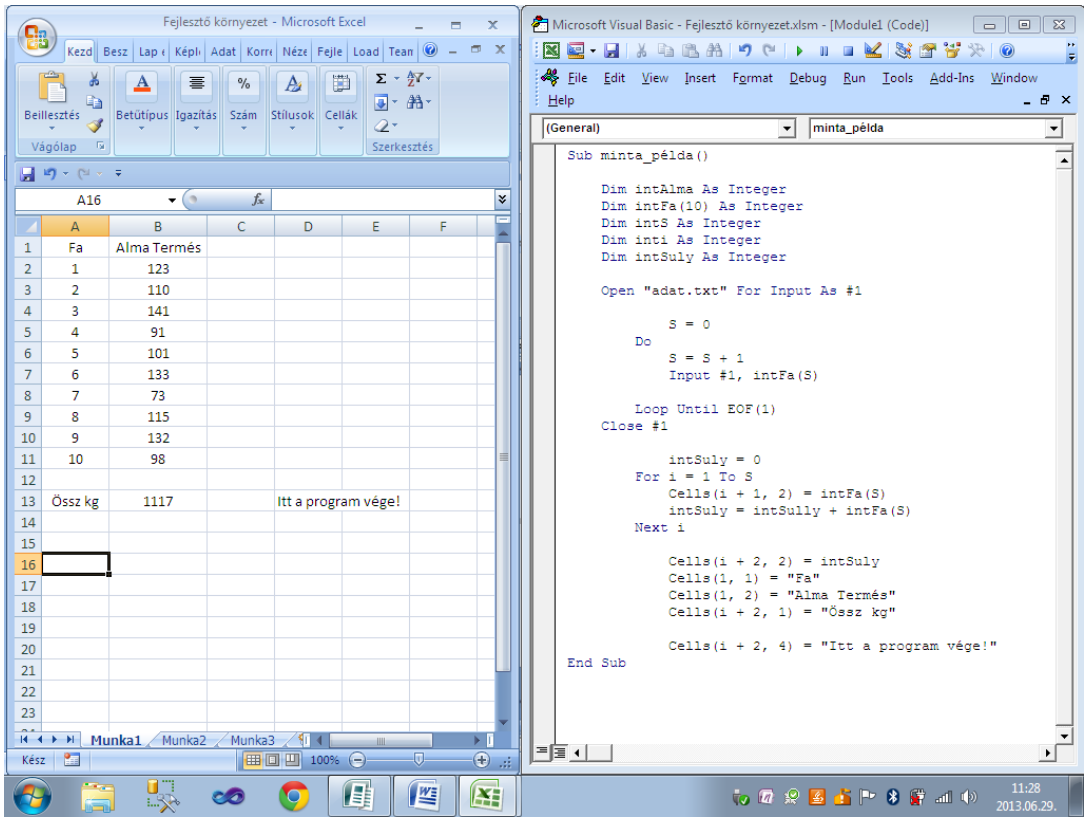
5.2 Visual Basic for Applications (VBA) fejlesztő környezete

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 egy sor 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 (Ablakok) operációs rendszer lehetővé teszi, a kétablakos programozást, ezért ez a legjobb kialakítás ahhoz, hogy maximálisan „élvezni lehessen” a szövegszerkesztő mögé integrált működését.

A baloldali ablak az **Excel**, a jobboldali ablak a **fejlesztő környezet**.



A jobboldali 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 Ciklus utasításokba rendezett értékadó parancsokat tartalmazó része található, melynek futásakor tölti ki a program a baloldali Excel munkafüzetet.

A programot alaposan szemlélve látható, hogy a program áttekinthetősége egyrészt szavak kék színű kiemelésének, másrészt a program szövegének blokkosításával (például a „Dim” utasítások egymás alatt vannak, nem össze-vissza a szövegben) és a leírt szavak balról történő, különböző mértékű behúzásának köszönhető.

5.3 A VBA fejlesztőrendszer fogalmai

5.3.1 Modul lap:

Modul lap megjelenése egy „Üres papírlap” látványát mutatja. Tudni kell azonban arról, hogy az ezen való írást egy olyan speciális szövegszerkesztő segíti, mely nemcsak a „Program írótól” fogad el írásutasításokat, hanem a Fejlesztő Környezet „eszközeként”, a mögé integrált Hibafelismerő programtól is.
Természetesen a Program író – 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, ezen Modul lapokon történik, (lásd, a fenti ábra Jobb ablaka). Tekintettel arra, hogy a ”Fejlesztő környezet és vezérlői eszközei” és így a Modul lap is Angol nyelvű programozású, ezért programírás is, az Angol ABC használatát követeli. A Modul lapra írt (alfa-numerikus) program leírást ezért, a VBA szabályai szerint és Angol ABC-vel kell írni.

Itt kell azért felhívni a figyelmet arra, hogy a Fejlesztő Környezet bal ablakába, az Excel – magyar nyelvezetű verzióját futtajuk, ezért itt a magyar írás használatos (Ékezetes betűk és pl. a „Tizedes” jel 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, és a „Tizedes vessző”)

5.3.2 Forrásnyelvű programleírás:

A Modul lapra, a VBA szabályai szerint a sorokba írt utasítássorozat alkotja a programleírást.
Ez ember számára olvasható (alfa-numerikus) program leírás **az alapja – „forrása”**, a fejlesztő környetzetbe integrált automatikus fordító programnak. (Compiler-nek) A fordító program az ember számára olvasható (alfa-numerikus) leírást fordítja, a számítógép számára értelmezhető Hexadecimális Gépikód sorozatokká.

Biztonsági figyelmeztetés! Itt kell felhívni a figyelmet arra, hogy a „programok védelme és értéke” ebben az ember számára olvasható (alfa-numerikus) program leírásban van, ezért ha védeni akarja valaki az „eltulajdonítástól, vagy illetéktelen felhasználástól”

5.3.3 Fordítóprogram, *(Compiler)*:

A fordítóprogram feladata, hogy Modul lapon írt és „készre nyilvánított” forrásprogramot, a számítógép számára értelmezhető Hexadecimális Gépikód 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 kódolta a forrásnyelvű szöveget, hanem egy közbenső nyelvre (p-kódra). 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 gép-független standarddá vált.)

5.3.4 Hibafelismerő program *(Debugging)*:

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

Itt kell megemlékeznünk Kemény Jánosról, mert munkásságának élvezői a mai Hallgatók is. *"Nem csak 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".* **Az első BASIC program 1964. május 1-jén futott.**

Ezen integrált Hibafelismerő program valósítja meg **Kemény János azon elhatározását is**:

„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 programozási nyelv legyen alkalmas eltérő célok kielégítésére.
- A nyelv kezdők számára is könnyű legyen.
- A magasabb szintű utasításokat csak később kelljen megtanulni.
- A használó és a gép között a nyelv legyen interaktív.
- Könnyen érthető hibajelzéseket adjon.
- A speciális gép-architektúra ismerete nélkül is lehessen használni.
- Óvja meg a használót a gép operációs rendszerének problémáitól.

A program tesztelése annak ellenőrzésére szolgál, hogy a program a tervezettnek megfelelően működik-e? A tesztelés többszintű, többfeladatú tevékenység, melyet ráadásul a programírás különböző fázisaiban kell végezni. A tesztelést, a Hibafelismerő vizsgálatokat, parancssorok irásának, 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, nyelv szabályainak be nem tartásából eredő vizsgálat, már parancssoronként aktivizálódik,
- értelmezhető jelzéseket ad a hibákról, de összetettebb nyelvi struktúrák hibafelismerésére nem mindig képesek,
- 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, melyek végrehajtására a program nincs felkészítve, ezért ezek a program azonnali befejezését eredményezik, s 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.

5.4 A Visual Basic programnyelve – Objektumorientált programnyelv.

A *Visual Basic-ben* fejlesztett alkalmazási program egymással kapcsolatban álló, egymásra ható objektumok rendszere.

Az egyes objektumoknak (miként az embereknek is**)** ***tulajdonságaik***, önálló ***cselekvéseik*** (- tevékenységeik; szaknyelvre fordítva: ***metódusaik)*** és a velük kapcsolatos ***eseményekre adott válaszaik*** (- *esemény-kezelő* ***eljárásaik)*** **vannak.**

A működő programok **tevékenységeinek tárgyai** mindig valamilyen **Objektumok.**

- Ezek lehetnek olyan elemiek, mint a gép memóriájának egy bitje, bájtja (byte),
- de lehetnek ezekből felépített, egyszerűbb vagy bonyolultabb struktúrák (számok, szövegek, …stb.).

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

5.4.1 Projekt programozás:

A legegyszerűbb, - a mindig szükséges *VB* futtató magot nem számítva - önmagukban is futásképes programok egy-egy *VB IDE* alkalmazásfejlesztési *projekt* eredményei. A *projekt* a fejlesztése folyamán keletkező információkat, eszközöket fájlokban tárolja. A fájloknak a fájlstruktúrában történő elhelyezéséről a programfejlesztő szabadon dönthet, azonban praktikus betartani az *egy projekt - egy könyvtár* elvet, a különböző fejlesztésekkel kapcsolatos információk jó áttekinthetősége érdekében. A Projrkt Könyvtárat, a VBA Fejlesztő felületén lehet láthatóvá tenni. Megjelenítése: Insert menü / Projrct Explorer választással lehetséges. A VBA felület bal oldalán nyílik meg, s láthatók benne könyvtárszerűen a tárolt objektumok.pl: Excel munkalapok, Excel Munkafüzet, és ha a projektban vannak Modul lapok és Formok.
Ebből az Explorerből is klikkeléssel megnyithatók a választott objektumok.

Hogy ne ússzunk teljesen a levegőben, *most nézzük meg a Basic* programnyelv *program-írás alapvető formai szabályait, és a két legfontosabb utasítását.*

Eseménykezelő Eljárások: Subrutin – Function, és ezekben használatos Értékadó utasítások és Metodusok
Megjegyzés: A további utasítások később részletesen vannak tárgyalva. Feltételes (~conditonal) utasítások –lsd. 5.5.9 alatt, Ciklusszervező utastások, - lsd. 5.5.10 alatt.

5.4.2 Eseménykezelő eljárások:

A programírás során egyéni – **felhasználói Eseménykezelő eljárásokat írunk**. Ezeket Eseménykezelő eljárásokat, Modul lapokra írt rutinokba, -Subrutin vagy Function rutinba kell irni. A rutinokat az Általános elnevezési szabály szerint el kell nevezni (lsd. 5.4.6), hogy hivatkozni lehessen rájuk. A Név után () zárojelben kell felsorolni, a másik rutinnak átadni szándékozott Változókat. Ezt az átadást „paraméter átadásnak” hívják, és a zárójel párt akkor is ki kell tenni (üresen), ha nincs paraméter átadás.

5.4.3 Az értékadó (~assignment) utasítás

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

Az értékadó utasítás (~assignment statement) műveleti jele (~operator) az értékadó-operátor (=). Az = egyenlőségjel tehát itt nem a matematikai egyenlőséget jelent, hanem az Értékadó Utasítás olyan „kitüntetett jele", mely szétválasztaja a az Utasítás Jobb és Bal oldalát. Ezért lehet *Szeparátor*nak is nevezni, mert ez – mintegy „elszeparálja" a jobb és bal oldalt.

Az utasítás pedig az alábbi formában írandó:

<objektum-tulajdonság \| változó> = <érték>*

amelyben

<objektum-tulajdonság>: egy objektum valamely módosítható tulajdonsága

<objektum-név>.<tulajdonság-név > formában megadva,

<érték>: a bal-oldalon álló szintaktikai egység *típusának megfelelő* (de arra legalábbis átkonvertálható) *kifejezés* kell, legyen (lényegében bármely *konstansokat, változókat, vagy objektum-tulajdonságokat* használó képlet).

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 adódjon a bal oldalon álló változónak vagy objektum-tulajdonságnak.

Attól, hogy egy konkrét értékadó utasítás baloldalán álló szintaktikai elem (változó vagy objektum-tulajdonság) csak szöveg befogadására alkalmas, a *VB* nyelv szabályai szerint megengedett az, hogy a jobb oldalon álló kifejezés numerikus értéket állítson elő, de az értéknek a szintaktikai elembe történő beírása előtt szöveggé konvertálódik.

5.4.4 Metódus végrehajtása ("hívása")

Az objektumokkal kapcsolatos minden tevékenységet, az objektum típusának megfelelő metódusok végzik. A metódusok végrehajtásának igénylése a programban

<objektum-név>.<metódus-név> <paraméter-lista> formában történik,

ahol a

<paraméter-lista> a metódus működéséhez szükséges paraméterek (ténylegesen: értékek) vesszővel elválasztott listája.

Megjegyzés: e szabályt kivételek "erősítik" (például: bár a *Form1.Show* utasítással - azaz a *Form1* formra alkalmazott *Show* metódussal - a form megjeleníthető, azonban ha csak a memóriába történő betöltését igényeljük, akkor ezt a *Load Form1 Basic* nyelvi utasítással kell megtenni, ami igencsak nem "metódushívásszerű").

5.4.5 A program - kódírás formai szabályai

A Visual Basic programírás fő követelménye, az átláthatóság! A funkcionálisan azonos parancsok elkülönülése, a „jobbra tartó írásmód”, amihez az üres sorok beszúrása és Tabulátor „erőteljes” használata ad lehetőséget. Az alábbi irányelveket javasolt betartani:

- Sub End Sub oszlopába semmit nem szabad írni.- kivétel a „Címke” és a „Komment”.
- Minden parancsot lehetőleg külön sorba kell írni. Amennyiben mégis több parancsot írunk ugyanabba a sorba, akkor a parancsokat kettősponttal válasszuk el egymástól.
- Összetett parancsok kulcs sorainak kezdő oszlopába pl.(For ...Next) (If... Else ...End If) (Do …Loop) nem szabad semmit írni, a közéjük kerülő utasításokat egy Tab-bal jobbra kell írni.
- Egymásba ágyazott, összetett parancsok írására is ugyan ez vonatkozik, (például 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 íródjanak.

```
Sub Bemutato_0()
End Sub
```

```
Sub Bemutato_1()

    Dim intX As Integer
    Dim sngY As Single

Elolrol:

'Bemutató példa
    intX = InputBox( ..... )
        GoTo Elolrol

End Sub
```

```
Sub Bemutato_2()

    Dim intX As Integer

    intX = InputBox( ..... )

    If intX > 10 Then
        cells(intX, 1) = intX
    Else
        cells(intX, 1) = intX
    End If
End Sub
```

```
Sub Bemutato_3()

    Dim intX As Integer
    Dim sngY As Single
    Dim strZ As String

    For  intX = 1 To 7
        cells(2,3) = intX
    Next intX

End Sub
```

```
Sub Bemutato_4()

    Dim intS As Integer
    Dim sngY As Single

        intS = 0
    Do
        intS = intS + 1

    Loop Until intS > 7

End Sub
```

- Az úgy nevezett: *white space* (azaz a szóköz -, a kód indentálásra szolgáló tabulátor és a komment jel – az Aposztrof jel) karakterek a kódban bárhol előfordulhatnak, kivéve a *Basic* kulcs szók- és operátorok-, valamint a program-változó- és. az objektum-azonosító nevek belsejét**.**
- Szóközöknek az utasításokon belüli írásával nem is érdemes nagyon bajlódni, mert a *Basic* fordítóprogram az utasítást a sor *Enter* billentyűvel történő lezárása után úgy is automatikusan átformálja. Ha a sorban olyan szintaktikai hiba lenne, amit a fordító képes "ott, helyben" felismerni, azt a hibaszínnel jelzi; de a helyesen írt, és felismert *Basic* ku1csszók is megkülönböztető színnel jelenítődnek meg. A *VB* szövegszerkesztője biztosítja még a változóneveknek az első előfordulásuknak megfelelő automatikus átformálását.
- 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.

5.4.6 Általános elnevezési szabályok:

A VBA programozásban létező objektumok és azonosítók (Változók, mint speciális objektumok) elnevezésére általános szabály van. A **Név**, egy Nagybetűvel kezdődő, legfeljebb 255 karakter hosszúságú szöveg, mely a második karakterétől kezdve számjegy, - vagy aláhúzás karakter (_).

Simonyi Károly javaslata az, hogy úgynevezett „Beszélő Neveket" alkalmazzon a programozó annak érdekében, hogy könnyebb legyen a programértés és ne „kódszavakat" kelljen memorizálni. A Nevek lehetnek összetettek – több szóból is álló kifelyezés is, de ilyenkor az „_" aláhúzás jellel egy taggá – Névvé, kell összefogni.
Simonyi Károly a Változókra is kiterjesztette a „beszélő Név" megadást még azzal, hogy a változó Tipusának 3 kisbetűs rövidítését, a nagybetűvel kezdődö név elé illesztette. Használata célszerű!!!

A nevek egyikében sem használhatók:

- a magyar ékezetes betűk, (Windows magyar nyelvű operációs rendszere esetén sem),
- VB kulcsszavak (VB parancsok, illetve VB belső függvények nevei),
- a változók adattípusára jellemző karakterek, szóköz és pont.

5.4.7 Az azonosítók használata, 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), ezen részek összességét az azonosító **hatáskörének** nevezik (lásd a „Visual Basic for Applications (VBA) fejlesztő környezete" fejezetben leírtakat). A hatáskör lehet:

- **globális**, az azonosító a program összes moduljának (a modulokban történik a programok írása) összes alprogramjában (eljárás, függvény) 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**.

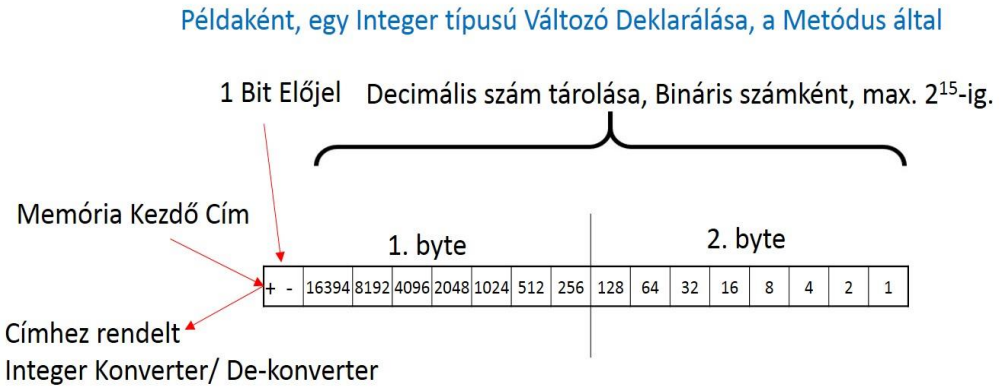
Az azonosító élettartama a program végrehajtásának azon periódusa, melyben az azonosító létezik, jól definiálható értékkel rendelkezik:

- **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.

5.5 A Visual Basic utasításai és parancsai

5.5.1 Változók és konstansok Deklarálása és használata

A változók és a konstansok a VBA program legegyszerűbb objektumai, mert csak egy meghatározott típusú adat (szám, szöveg, logikai érték) tárolására képesek. Ezen információn kívül nincs egyéb tulajdonságuk és önálló cselekvéseik (szaknyelvre fordítva metódusok) sem.
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 elején, - a Subrutin elején kell felsorolni azért, hogy a Deklarációk után következő Adatbevitel – Adatfeltöltés programsorainál, már kész tárolóhelyekként szolgálhassanak.
A változó-deklarációs utasításának feladata: - az Általános elnevezési szabályok által elnevezett Változónak, a kulcsszavak által meghatározott élettartamú – hatáskörű és adattípusú tárolóhelyt hozzon létre a nyelv fordítóprogramja, az élettartamukra rendelt memóriaterületen.



A Dim kulcsszó behív egy Meódust, mely a Változó nevéhez hozzárendel egy Kezdő memória címet, majd az Integer számábrázoláshoz szükséges, egymás utáni 2 Byte- Bítjeit 0-ba állítja. Ezután a Kezdő címhez hozzárandel egy olyan Konvertert/De-Konvertert, mely képes forgatni, az Integer tipusi Változónak, a Modul lapon, vagy Excel cellába írt Decimális (10-es számrendszerű) számait, a Memória Bináris kódólásu számábrázolásra között.

Összefoglalva, és általánosan elmondva, tudni kell, hogy az Excel cellákból vagy a Modullapon írt programból származó Alfa – Numerikus adatokkal a számítógép, csak ezek 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 le, a kijelölt memória területeken. Viszont az Excel celákba való megjelenítéshez de-konverziót, visszakonvertálást kell végrehajtani, az olvasható megjelenítás érdekében. E konverziók helyes működése érdekében szükséges tehát előre megadni az egyes adatok „tárhelyeire” vonatkozó „típus” utasításokat is.

5.5.1.1 Változók – Konstansok, Alapvető adattípusai

Az adattípus az állandók és a változók által hordozott információ jellegét és lehetséges értéktartományát határozz a meg. A adattípusok az információ jellege alapján lehet: numerikus, szöveges, logikai, dátum-idő 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 a csupa 0 bitekből álló-, a True pedig a legalább egy bitjén nem 0 értékű duplabájtnak felel meg);
- **Integer** (egész): 2 bájton tárol egy [-32768 ≤ x ≤ 32767] intervallumbeli 10-es számrendszerbeli egész számot;
- **Long** (egész): 4 bájton tárol egy [-2147483648 ≤ x ≤ 2147483647] intervallumbeli 10-es számrend-szerbeli egész számot (kódolás, stb. mint előbb);
- **Single** 4 bájton tárol egy (-precision floating-point; egyszeres pontosságú lebegőpontos): az IEEE 32-bites szabvány szerint tárolja az 1.401298 E-45 ≤ x abszolút értéke ≤ 3.402823 E38 feltételnek eleget tevő 10-es számrendszerbeli valós számot (Iassú lebegőpontos aritmetika);
- **Double** 8 bájton tárol egy (dupla pontosságú lebegőpontos): az IEEE 64-bites szabványa szerint tárolja a 4.94065645841247 E-324 ≤ x abszolút értéke ≤ 1.79769313486232 E308 feltételnek eleget tevő 10-es számrendszerbeli valós számot (még lassúbb aritmetikájú);
- **Currency** (- pénz): 8 bájton, skálázott egész számként kódolva tárol egy -922337203685477.5808, ≤ x ≤ +922337203685477.5807 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;
- **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; kicsit lassúbb kezelést biztosít;
- **Variant**: a variant adattípus tárolhat mind számot, mind szöveget, de memóriaigénye jóval nagyobb (szám: 16 bájt, szöveg: 22 bájtos adatfejet + szöveg hossza)

Az alapvető adattípusú változók deklarálásának szintaxisa:

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

Simonyi Károly szerinti összetett Változónév használata javasolt. A Nagybetűvel kezdödö Változónév elé, 3 kisbetűs a Típusra jellemző rövidítést kell írni. Így a program írása és futtatása közben is mindig könnyű nyomon követni, hogy melyik Változóban, milyen adatok lehetségesek.

5.5.1.1.1 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 alábbi alapvető adattípusok alapján történhet (lásd az „Alapvető adattípusok” fejezetet)

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
```

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. Ennek érdekében 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, vesszővel elválasztott két értéket kell írni.

Egy-indexes (Sor-index), vagy „Egy-dimenziós” 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
```

Két-indexes (Sor-index, Oszlop-index) vagy „Két-dimenziós” 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
```

5.5.1.1.2 Egy- Indexes - 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.

5.5.1.1.3 Konstans azonosítók

A konstans 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”, szoveg2 as string = "nincs"
```

5.5.1.1.4 Az alapvető és gyakrabban használt adattípusok rövidítései

Az alapvető és gyakrabban használt adattípusokat különböző jelekkel is meg lehet adni:

Adattípus:	Integer	Long	Single	Double	String
Jel:	%	&	!	#	$

Természetesen az adattípusok jelei a változók deklarálásánál is használhatóak:

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

5.5.1.2 Felhasználói adattípusok

A Basic legpraktikusabb képességei közé tartozik, hogy lehetőség van a felhasználó által definiált adattípusok (user defined type) létrehozására. A felhasználói típus (melyet gyakran neveznek struktúrának is) egy összetett adattípus, ami azt jelenti, hogy kettő vagy 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. 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 VBA fejlesztőrendszer bármely más beépített adattípusát.

Ilyen Felhasználói struktúrát, csak modulszintű deklarációban lehet létrehozni, úgy a **Public** kulcsszót használjuk. Ekkor a változó hatóköre az egész projekt lesz, az összes modulból használhatjuk az ilyen változókat. Ugyanilyen jelentéssel használhatjuk a **Global** kulcsszót is:
Nézzünk egy példát: - Az előbb létrehozott **Személy** 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”`

5.5.1.2.1 Felhasználói adattípusban, a Tömbök használata.

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)-as 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 a

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
```

5.5.2 Értékadó parancssorok utasításában használt műveletek és függvények:

5.5.2.1.1 Aritmetikai műveletek:

- összeadás, jele: +,
- kivonás, jele: -,
- szorzás, jele: *,
- osztás, jele: /,
- egészszámosztás: 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,

5.5.2.1.2 Relációs műveletek:

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

5.5.2.1.3 Szövegkezelő művelet:

A szövegek, illetve a szöveg és a nem szöveges adat összefűzésére szolgál (a nem szöveges adat előzőleg szövegessé konvertálódik, jele: &.

5.5.2.1.4 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(<radián>): 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 alapu logaritmust számoltatni kell. pl: $Log_{(10)}$ (intX) = Log(intX) / Log(10)
- Sqr(<nem-negatív>): a <nem-negatív> szám négyzetgyökét adja meg,
- Rnd: véletlen szám generálása 0 és 1 közötti értékben,

5.5.2.1.5 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.

Mintapéldák:

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

5.5.2.1.6 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" eredménye akkor igaz, ha „x” értéke 3 és 7 közé esik.
- Not: tagad: logikai változó értékét változtatja az ellentétére, ha „x” értéke „true”, akkor Not x értéke „false” lesz.

5.5.3 Az „Option …” utasítások

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

- **Option Explicit**: az Option Explicit utasítás használata esetén csak olyan változókat használhat a program, amelyeket az „Alapvető adattípusok” alfejezetben leírtaknak megfelelően a programozó deklarált a változó használata előtt. Enélkül a parancs nélkül a Visual Basic megengedi, hogy a változót (variant adattípussal) a program automatikusan hozza létre a változó első alkalmazása előtt (implicit deklaráció). Az Option Explicit használatával elkerülhető hogy egy változónév elírásával akaratunkon kívül új változót hozzunk létre és az értéket is kapjon.
- **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**: az Option Compare Text alkalmazása esetén a Visual Basic nem tesz különbséget az adott karakter „kisbetűs” és „nagybetűs” változata között. String típusú változók vizsgálatának esetén használható.

5.5.4 Műveletek szöveges változóval

A szöveges 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 5 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 szöveg vagy szövegváltozó szövegéből, a szöveg elejétől a <karakter_szám>-nak megfelelő számú karaktertől. az <utána_szám>-nak megfelelő számú karaktert ad vissza a <szöveg_változó>-ba.
- Szövegek „összeadása” az „&” karakterrel történik. Szintaxisa:

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

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

Feladat: a Program olassa be a Vegyészmérnöki Kar szöveget, és a megfelelő utasításokkal elemeze 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.

Program **Megjegyzés**

```
Sub szoveg()

Dim strNev As String, intX%, strNevresz$

strNev = "Vegyészmérnöki Kar"
Cells(3, 2) = strNev

intX = Len(strNev)                                  Szöveghossza - karakterszám
Cells(3, 4) = intX

strnevresz = Left(strNev, 3)                        Szöveg eleje
Cells(3, 5) = strnevresz

strnevresz = Right(strNev, 3)                       Szöveg vége
Cells(3, 6) = strnevresz

strnevresz = Mid(strNev, 5, 3)                      Szöveg közepe
Cells(3, 7) = strnevresz

strnevresz = Left(strNev, 7) & " " & Right(strNev, 3)   Szöveg összefűzése
Cells(5, 2) = strnevresz

End Sub
```

	A	B	C	D	E	F	G
1		Szöveg		Hossz	Eleje	Vége	Közepe
2							
3		Vegyészmérnöki Kar		18	Veg	Kar	ész
4							
5		Vegyész Kar					
6							
7							

5.5.5 A Like parancs :

Segítségével meg tudjuk mondani, hogy a szöveg tartalmazza vagy sem a programozó által megadott karaktersort. Ha „Eredmeny” változó boolean típusú, akkor az

Eredmeny = Opera Like "O*"

parancsor után „Eredmeny” változó értéke true lesz, mert az Opera szó valóban „O” betűvel kezdődik. Az alábbi táblázat a Like parancs alkalmazására mutat néhány példát.

Összehasonlítás	Eredmény	Megjegyzés
"Motor" Like "M*"	True	A * egy vagy több karaktert helyettesít.
"Motor" Like "?o*"	True	A ? egy karaktert helyettesít.
"3-as" Like "#-as"	True	A # egy számkaraktert helyettesít
"u-as" Like "#-as"	False	
"3-as" Like "?-as"	True	
"u-as" Like "?-as"	True	
"Jani" Like "Jan[ió]"	True	A [ió] az i-ó tartományban egy karaktert helyettesít.
"Jani" Like "[A-M]*"	True	
"Jani" Like "[!A-M]*"	False	A ! kizárja az A-M tartomány karaktereit.

5.5.6 Cellaparancsok

5.5.6.1 Cella használata írásra - olvasásra

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

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

ahol 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.

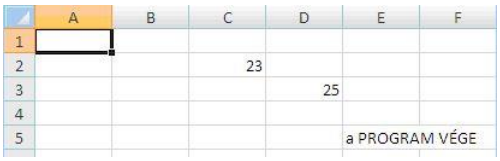
Megjegyzés:

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 képzet- egész számok (pl. Cells (x + 4, y + i).

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

Program	**Megjegyzés**
Sub cella_muveletek()	
Dim intAdat As Integer Dim sngEredmeny As Single Dim intX As Integer Dim intY As Integer	
Cells(2, 3) = 23	Kiírás cellába "direkt címzéssel”.
intX = 3 intY = 4 Cells(intX, intY) = 25	Kiírás cellába "indirekt címzéssel".
intAdat = Cells(2, 3)	Beolvasás cellából.
sngEredmeny = Cells(3, 4) / intX	Beolvasás cellából művelettel.
Cells(5, 5) = "a PROGRAM VÉGE"	Ajánlott a PROGRAM VÉGE megjelenítése.
End Sub	

5.5.6.2 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 (balfelső sarok, az „A” oszlop fejléce mellett és az „1” sor felett), majd a billentyűzeten a „Delete” gombot lenyomva,
- 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 első esetben az Excel aktív munkalapjáról tudunk adatokat törölni, addig a 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

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

5.5.7 Adatbevitel és adatkivitel objektumokkal

5.5.7.1 InputBox - A VBA program adatbeviteli objektuma

Az InputBox objektum alábbi lehetséges tulajdonságai közül 3 tulajdonságát javasolt megadni a zárójelben:

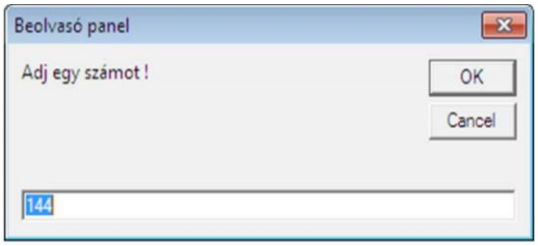
```
InputBox(Prompt, [Title], [Default], [XPos], [YPos], [HelpFile], [Context]) As String
```

- **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 az

```
x = InputBox("Adj egy számot!","Beolvasó panel",144)
```

parancsot adjuk ki, akkor a jobboldalon lévő ablak tűnik fel.



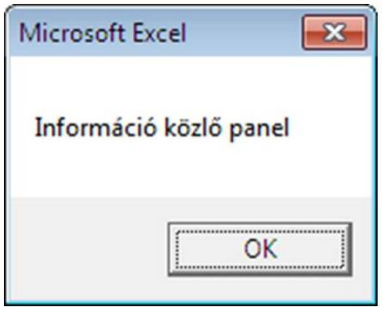
5.5.7.2 MsgBox - A VBA program adatkiviteli objektuma

Az MsgBox objektumnak minimum egy tulajdonságát, a Prompt (közlés) tulajdonságát meg kell adni, mely a felhasználó részére szolgáltat információt és a szürke mezőben jelenik meg.

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

```
MsgBox "Információközlő panel"
```

parancs kiadása után a jobboldalon lévő ablak jelenik meg.



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

```
MsgBox "x = " & x
```

5.5.8 Adatbevitel és adatkivitel *.txt fájl segitségével

A VBA programba text típusú fájlokból (általában *.txt fájlok) lehet adatokat beolvasni, illetve ilyen fájlokba lehet 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 ismeri fel a VBA.
A VBA programban az Open ….Input...Close parancsutasítás látja el a feladatot úgy, hogy az „Input" parancs kijelölt *.txt fájlból, balról - jobbra haladva „karakterláncokat" olvas be a megadott változóba. Egy „karakterláncnak" tekinti az „Input" utasítás azt a szöveg vagy számkarakter sorozatot, melyet:

- szám, illetve idéző jelek között lévő szöveg esetén beolvasása esetén vessző, pontosvessző, space, tab, enter, EOF vezérlő karakter,
- míg idézőjelek nélkül lévő szöveg esetében enter, EOF vezérlő karakter zár le.

Az Open … Close utasítás együttes szintaktikája:

```
Open <fájlnév.txt> For <alkalmazás - Input, Output vagy Append> As # <virtuális memórianév>
Input #<a virtuális memória neve>, <eltárolás helye - változója>
```

illetve fájlba történő írás esetén:

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

Végezetül a fájlt be kell zárni:

```
Close #<a virtuális memória neve>
```

Az „Open" megnyitja a <fájlnév.txt> -t beolvasásra (Input), kiírásra (Output), vagy hozzáfűzésre (Append) és a beolvasás vezérlését átadja (As) a megnevezett virtuális memória résznek.
Az „Input #" a virtuális-memória vezérlése mellett kiolvassa a megnyitott text fájlból, a kért „karakterláncot" és a beolvasott karakterláncot eltárolja a kijelölt változóba.
A „Write #" 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.
A „Close # <a virtuális memória neve>" bezárja a megnyitott text fájlt.
A beolvasó 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.
Amennyiben a Szamsor.txt fájlt szeretnénk megnyitni, akkor az alábbi paranccsal tehető meg:

```
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 lementésre. Amennyiben 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
```

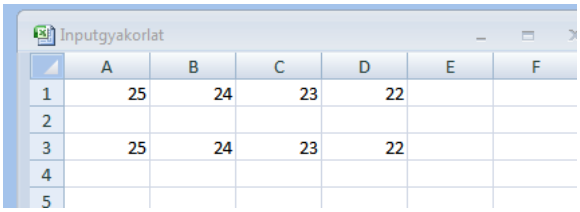
Ugyanakkor sokszor előfordulhat, hogy a felhasználónak kell megkeresnie a beolvasandó fájlt, vagy meghatároznia mentés helyét, amit az

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

```
Open Application.GetSaveAsFilename as #1
```

parancsok futásakor tehet meg a Windows alapértelmezett „Megnyitás” és „Mentés másként” ablakai segítségével.
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, s és utána a Cells paranccsal kiírja az Excel munkalapra.

5.5.9 Elágazó utasítások. Feltételes és feltétel nélküli utasítások

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

5.5.9.1 Go to 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.

5.5.9.2 If … then 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. Jellemző példája az If …. then utasítás, melynek szintaxisa:

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

Ha a <feltétel-1> = Igaz, 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-al.

A beolvasást követő adatfeldolgozásban értékelje a kapott számokat.

- ha a szám 0, akkor nem kell több számot kérni, hanem írjon „értesítést cellába” és lépjen ki .
- ha a szám nagyobb mint 10, akkor írjon „értesítést cellába” és kérjen új számot.
- ha a szám megfelelő, akkor írjon „értesítést cellába” – abba a sorba, amekkora a szám, de értékelje azt is, hogy a szám, nagyobb mint 7, illetve nagyobb mint 3, vagy kisebb mint 3. és lépjen ki.
- Ezután kérjen új beplvasást.

Program

```
Sub felteteles_elagazo()

Dim intSzam As Integer

ujszam:
intSzam = InputBox("intSzam = ? (0-10)")

If intSzam = 0 Then                                  Alsó érték ellenőrzése
   Cells(11, 1) = "0 bevitel, nincs több beolvasás"
   GoTo vege
End If

If intSzam > 10 Then                                 Felső érték ellenőrzése
   Cells(11, 1) = "10-nél nagyobb. új beolvasás"
   GoTo ujszam
End If

If intSzam > 7 Then                                  Érték szerinti kiírás
   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                                          Ugró utasítás

vege:

End Sub
```

Megjegyzés

	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		

5.5.9.3 Select Case … End Select elágazó parancs

If …. then 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 előző 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 bevitel, nincs több beolvasás"
   GoTo vege
Case Is > 10
   Cells(11, 1) = "10-nél nagyobb. 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		

5.5.10 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

5.5.10.1 For … Next ciklus

A For … Next ciklusutasítás a ciklusmagot ismétli egy numerikus típusú változónak a kezdeti értékétől a végső értékéig a <lépésköz> által definiált változás mellett. A For … Next parancs 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 értéknek>-nek, míg értelemszerűen negatív <lépésköz> esetén a <kezdeti érték>-nek kell a nagyobbnak lenni. A Step <lépésköz> elhagyható, ebben az esetben +1 lesz az érvényes lépésköz.

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

5.5.10.2 Do … Loop ciklus

A Do … Loop 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 addíg olvassák be 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 Mintaprogarm készült,
- Sub do_ciklus_1() , mely az Until Meódust használja a függvény-vizsgálatra, és a feladatot egymás után Hátul és Elöl-tesztelős megoldással is megolja a feladatot.
- Sub do_ciklus_2() , mely az While Meódust használja a függvény-vizsgálatra, és a feladatot egymás után Hátul és Elöl-tesztelős megoldással is megolja a feladatot.

Megjegyzés:

- A Do … Loop ciklusban nincs beépített ciklus lépésszámláló, ezért 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 elött kell egy Értékad parancs, melyben a Függvényben vizsgált Változót olyan értékre állítja, mellyel biztos, hogy először belép a ciklusba a program.

Program **Megjegyzés**

```
Sub do_ciklus_1()
Dim intX As Integer, intS%

intS = 0                                          a ciklus lépésszámlálója
Do                                                hátul tesztelő Until ciklus
   intS = intS + 1
   intX = InputBox("x=?", "Adatbekérés", 11)

   Cells(intS, 1) = intX
   Cells(intS, 2) = intX ^ 2
Loop Until intX = 0
```

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

```
intS = 0                                          a ciklus lépésszámlálója
intX = 12                                         x változó kezdő értéke

Do Until intX = 0                                 elől tesztelő Until ciklus
   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                                                hátul tesztelő While ciklus
   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                                      az x változó kezdő értéke

Do While intX <> 0                                elől tesztelő While ciklus
   intS = intS + 1
   intX = InputBox("x= ?", "Adatkérés", 11)
   Cells(intS, 4) = intX
   Cells(intS, 5) = intX ^ 2
Loop

End Sub
```

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

5.5.11 Programstrukturáló utasítások

A program strukturálásának alapvető célja, hogy a program fizikailag és/vagy logikailag szétbontható legyen apróbb, 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 olyan „Hívó" Subrutin, amely a Sub neve után üres zárójelpár van. (nem paraméterezett), és van egy olyan „Fogadó" rutin, - Sub, vagy Function rutin, melyek nevei utáni zárójelpárban, Változó deklarációk, - vesszővel elválasztva, 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 ez utáni Adatfeldolgozási részben lehet elhelyezni az egyes jól elhatárolódó feladatokat, másik rutinokkal való elvégeztetéséhez szükséges „Hívó" utasításokat. Ezek a „Hívó" utasítások 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 „Átadni kívánt" változókat. Az ide felsorolt Változóknak mind deklarálva kell, hogy legyenek, és a „Hívás" sorhoz érve „***Aktuális*** értékük-nek" is kell lenni. Éppen ezért, ebben a zárójelben felsorolt Változókat, „***Aktuális*** paraméterek" –nek nevezik, 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 rutint, Sub vagy Function, majd annak deklarációs részében levő ***Formális*** paraméterknek **„Átadja"** a Hivó oldali ***Aktuális*** paramétereket.
2. Ezzel a Fogadó oldali rutin számára így már ***Aktuálisak*** lesznek az addig csak ***Formai*** paraméterek, és ezekkel így az eljárás-törzsében levő parancsutasításokban el tudja végezni a programozott műveleteket.
3. A Fogadó rutin az eljárás-törzsében 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.

5.5.11.1 Elméleti – Működési magyarázat.

A paraméter ***Átadás*** metódus,, valójában paraméter ***„Átvétel”***-t valósít meg, mivel az ***Adatcsere metódusa az Átvételi*** deklarációban van megadva. Kétféle Átadást létezik, az ***Érték szerint*** és ***Cím szerint,*** mely 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.

5.5.11.1.1 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 memória- területekre). Ily módon az eljárás-törzs utasításaiban levő *Formális*-paraméter hivatkozások ténylegesen a megfelelő *Aktuális* paraméter-értékek másolataira vonatkoznak, ami 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 érték szerinti* paraméter-átadási mód esetén.

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.

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

A ByRef kulcsszó specifikálásakor *(vagy egyik kulcsszót sem megadva, hiszen ez az átadás alapértelmezett módja)*: nem értékelődik ki a hívásbeli ***Aktuális*** paraméter (amely 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, mely 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, mint egy „***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.

A program visszaadja vezérlést a Hívó Sub, hívó parancssorához. Ezután a Hívó Sub-ban folytatódik a feladat végrehajtás, az ***Aktualizált*** paraméterekkel.

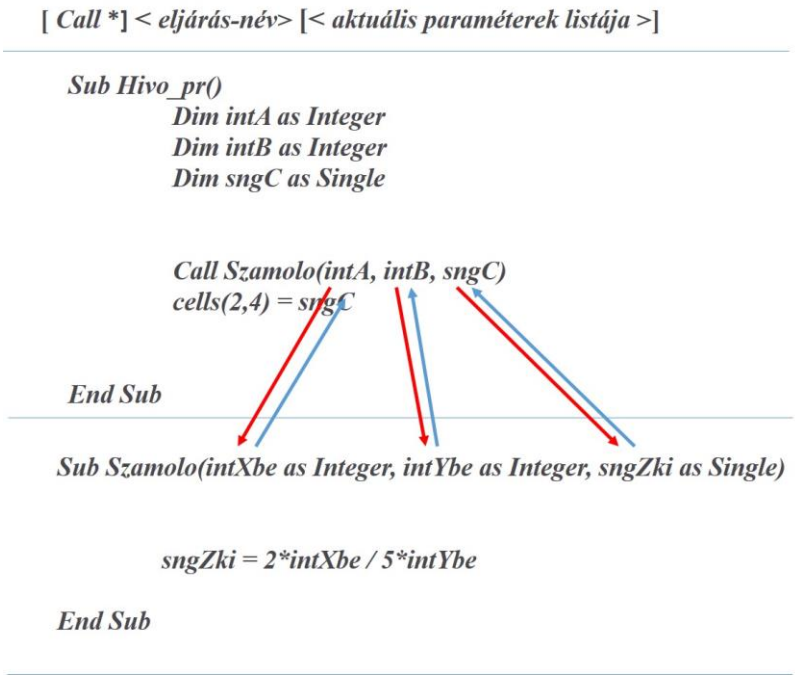
5.5.11.1.3 Paraméter átadás szabályai.

- 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 Projekten belül több helyről is meg lehet hívni. A különböző hívások esetén, úgysem lehet betartani a „Név-egyeztetést". Célszerű, úgynevezett „Beszélő" neveket adni azáltal, hogy –be vagy -ki jelzővel kiegészíteni a paraméter neveket, ami által követhetőbb a „Bejövő" és a „Kimenő" paraméter.)*
- Az „aktuális" és „formális" paraméter lista megfelelő paramétereinek, kompatibilisnek *(egymásba átkonvertálhatónak)* kell lenni. Ez azt jelenti, hogy például egy integer típusú aktuális paraméter átadhatja az értékét egy Integer, vagy Long típusú formális paraméternek *(de hibát okoz, ha a formális paraméter byte típusú és az átadandó érték nagyobb, mint 255)*.
-

5.5.11.1.4 Paraméterezett Szubrutin, vagy Funtion? Mikor – Mit?

Mindkét Eljárás paraméter- Átadáson nyugszik, és mindkettőnek az Aktuális és Formális paraméterek listái is azonos felépítésűek. Mégis van egy lényeges különbség az alkalmazásban, ami miatt nem lehetnek „csere- szabatosak„.

A) A paraméterezett Sub hívása esetében, a Hívó oldali () zárójelben felsorolt valamennyi Aktuális paraméter (Változó) Átadásra kerül a fogadó oldali Formális paraméterek aktivizálása érdekében, majd az eljárás-törzs parancsainak lefutása után, valamennyi paraméter értéke visszaíródik - felülíródik a hívó Aktuális paraméterekbe.



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

A Hívó_pr() nevű paraméternélküli Subrutin sorai láthatók, Deklarációkkal.

Sub Szamolo(……) rutin hívása látható az Aktuális paraméterek átadása, (⟶)

A visszakapott sngC változó felhasználása.

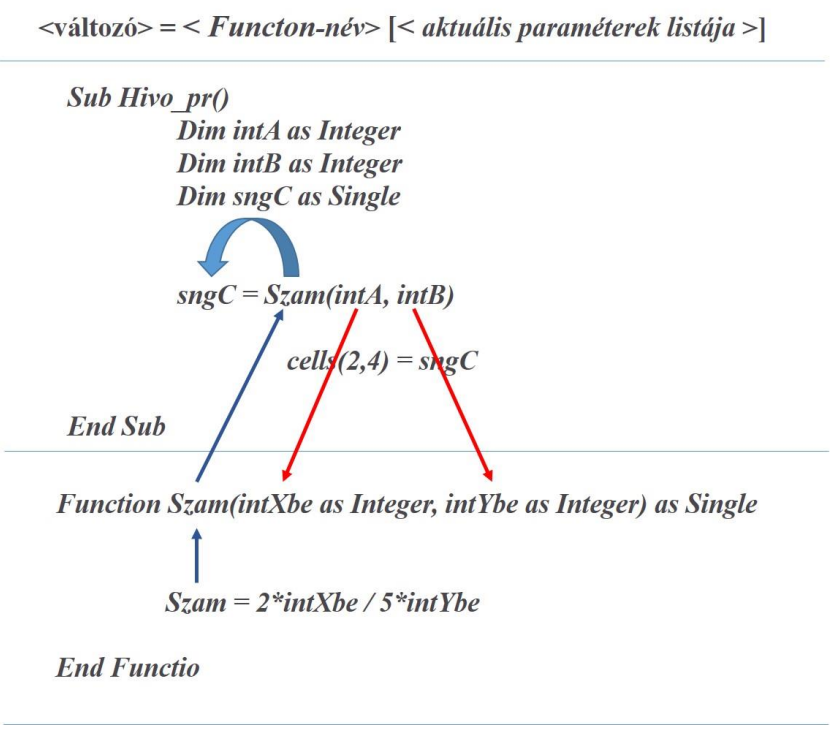
A fogadó Sub Szamolo(……) látható, a zárójelben a ***Formális*** paraméterekkel.

Az Eljárás-törzs számoló feladat látható.

Az End Sub végrehajtása a paraméter visszaírása. (⟵)

B) A Function hívása esetén is, a Hívó oldali ( ) zárójelben felsorolt valamennyi ***Aktuális*** paraméter (Változo) Átadásra kerül a fogadó oldali ***Formális*** paraméterek aktivizálása érdekében. De **Function** eljárás-törzs parancsainak lefutása után, a kapott paraméterekből kiszámolt „Egyetlen érték" íródik csak vissza a hívó oldali parancssorba. Ez úgy lehetséges, hogy az Aktuális paraméterek átadásával egyidőben, a Function „neve", a deklarációs zárójel után írt Változó típus deklaráció hatására, (pl. as Double) egy teljes értékű Változóvá deklarálódik és így képes lesz érték felvételére. A Function, eljárás-törzs 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 Aktulális paraméter lista zárójele előtti Névbe, mint Változóba írodik be.

- a hívó Aktuális paraméterekbe, semmi sem íródik vissza .



Ebben a sorban a Function rutin hívásának Szintaxisa látható.

A Hivo_pr() nevű paraméternélküli Subrutin sorai láthatók, Deklarációkkal.

Sub Szamolo(……) rutin hívása látható az Aktuális paraméterek átadása, (piros →)

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

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

Az Eljárás-törzs számoló feladat látható.

Az End Function végrehajtása a paraméter visszaírása. (kék ←)

Tehát, Mikor – Mit kell választani?

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

B) Ha az Aktuális praméterekből feldolgozásából – egyetlen adat keletkezik, *(pl. egy Függvény: (m = a*x^2 + b*x 3 c) m = értéke x helyen)*, akkor a **Function rutin** megvalósítása a megoldás.

5.5.11.2 Eljárás (Paraméteres Szubrutin)

Olyan önálló program, mely hívása céljából saját azonosító névvel és az információcsere céljából kapcsolódási felülettel rendelkezik. Az eljárás specifikálása bővebben *(lásd: 5.5.11.1.4. A)– 46. old)*

Az Eljárást (Paraméteres Szubrutin)-t meghívni, egy paraméter nélküli – de teljes értékű Szubrutinból lehetséges, s tartalmaz Deklarációkat, Adatbeviteleket, Eljárás-törzs programsorokat. Ennek az Eljárás-törzs 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. „Formálisak", mert tényleges változókká csak az eljárás meghívásakor válnak.

- A paraméterátadás szabályait a 46. old.-on tárgyaltuk. *(lásd: 5.5.11.1.3 – 46. old.).*
- E paraméter átadási mód, a címszerinti paraméter-átadás. *(lásd: 5.5.11.1.2 – 45. old).*

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

Sub Sokadik(intXbe%, intYki As Long)
Dim intI As Integer

intYki = 1
For intI = 1 To intXbe
   intYki = intYki * intXbe
Next intI

End Sub
```

Megjegyzés

	A	B
1	x	y
2	5	3125

Call hívás parancssor

A For ciklus helyett az
intYki = intXbe ^ intXbe
parancs is használható.

5.5.11.3 Függvény (Function)

Az Function rutint meghívni egy paraméter nélküli – de teljes értékű – Szubrutinból lehetséges, melynek eljárástörzsében van a Function hívósor. Az eljárás specifikálása bővebben *(lásd: 5.5.11.1.4 B) – 47. old)*

A Sub eljárásban levő Function „eljáráshívó” szintaxisa:

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

Az eljáráshoz képest azzal a két különbséggel rendelkezik, hogy 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 paraméterátadás szabályait a 46. old.-on tárgyaltuk. *(lásd: 5.5.11.1.3 – 46. old.).*
- E paraméter átadási mód a címszerinti paraméter-átadás. *(lásd: 5.5.11.1.2 – 45. old).*

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_példa()

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

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
```

Megjegyzés

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

Skalárszorzat számolása

„a” vektorhossz számolása

„b” vektorhossz számolása

A program a futásakor az „=” jeltől jobbra levő kifejezést hajtja végre. A fordítóprogram konstatálja, 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.

6 Feladatok

6.1 Programozási mintagyakorlatok alapvető algoritmusokkal

6.1.1 „n” elemű numerikus tömb elemeinek összege

Program

```
Sub osszeg()
Dim sngTomb(20) As Single
Dim intN As Integer
Dim intSum As Single
Dim intI As Integer

intN = 7
intSum = 0

For intI = 1 To intN
   sngTomb(intI) = Cells(1, intI)
   intSum = intSum + sngTomb(intI)
Next intI

Cells(1, intN + 1) = intSum
Cells(2, 1) = "n tömb elemeinek összege"
End Sub
```

Megjegyzés

Tömb elemeinek összege

	A	B	C	D	E	F	G	H
1	1	2	3	4	5	6	7	28
2	n tömb elemeinek Összege							

tömb beolvasása cellákból
egydimenziós tömb elemeinek összegzése

6.1.2 n x m elemű tömb elemeinek összege (összes, - oszlop, - sor, - átló)

Program

```
Sub osszes_osszeg()
Dim sngTomb(20, 20) As Single
Dim intN%, intM%, intI%, intJ%
Dim intSum!

intN = 5: intM = 7
intSum = 0

For intI = 1 To intN
   For intJ = 1 To intM
   sngTomb(intI, intJ) = Cells(intI, intJ)
   intSum = intSum + sngTomb(intI, intJ)
   Next intJ
Next intI

Cells(intN + 2, 1) = "n tömb elemeinek Összege"
Cells(intN + 1, intM + 1) = intSum

End Sub
```

Megjegyzés

Tömb elemeinek összege

	A	B	C	D	E	F	G	H
1	1	2	3	4	5	6	7	
2	1	2	3	4	5	6	7	
3	1	2	3	4	5	6	7	
4	1	2	3	4	5	6	7	
5	1	2	3	4	5	6	7	
6								140
7	n tömb elemeinek Összege							

a tömb beolvasása cellákból
a tömbelemek összegének számolása

```
Sub oszlop_osszeg()
Dim sngTomb(20, 20) As Single
Dim intN%, intM%, intI%, intJ%
Dim intSum_o(20) As Single

intN = 5: intM = 7

For intI = 1 To intM
   intSum_o(intI) = 0
   For intJ = 1 To intN
   sngTomb(intJ, intI) = Cells(intJ, intI)
   intSum_o(intI) = intSum_o(intI) + sngTomb(intJ, intI)
   Next intJ
   Cells(intN + 1, intI) = intSum_o(intI)
Next intI

Cells(intN + 2, 1) = "az oszlopok elemeinek összege"

End Sub
```

tömboszlopok elemeinek összege

	A	B	C	D	E	F	G
1	1	2	3	4	5	6	7
2	1	2	3	4	5	6	7
3	1	2	3	4	5	6	7
4	1	2	3	4	5	6	7
5	1	2	3	4	5	6	7
6	5	10	15	20	25	30	35
7	a oszlopok elemeinek Összege						

oszlopösszegek nullázása

a tömb beolvasása cellákból

```
Sub sor_osszeg()
Dim sngTomb(20, 20) As Single
Dim intSum_s(20) As Integer
Dim intN%, intM%, intI%, intJ%

intN = 5
intM = 7

For intI = 1 To intN
   For intJ = 1 To intM
   sngTomb(intI, intJ) = Cells(intI, intJ)
   intSum_s(intI) = intSum_s(intI) + sngTomb(intI, intJ)
   Next intJ
   Cells(intI, intM + 1) = intSum_s(intI)
Next intI

Cells(intN + 2, 1) = "a sorok elemeinek összege"

End Sub
```

a tömbsorok elemeinek összege

	A	B	C	D	E	F	G	H
1	1	2	3	4	5	6	7	28
2	1	2	3	4	5	6	7	28
3	1	2	3	4	5	6	7	28
4	1	2	3	4	5	6	7	28
5	1	2	3	4	5	6	7	28
6								
7	a sorok elemeinek Összege							

a tömb beolvasása cellákból
a tömb sorösszegeinek számolása

```
Sub atlo_osszeg()
Dim intSum_atlo as Integer
Dim intN%, intM%, intI%, intJ%

intN = 7: intSum_atlo = 0

For intI = 1 To intN
   intSum_atlo = intSum_atlo + Cells (intI, IntI)
Next intI

Cells(intN +1, intN+ 1) = intSum_atlo
Cells(intN + 2,1) = "Tömb bal átlója elemeinek összege"
End Sub
```

tömb bal felsőből induló elemeinek összege

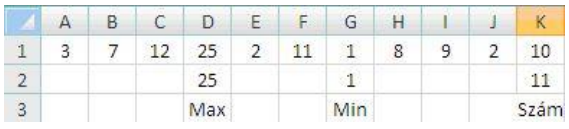
	A	B	C	D	E	F	G	H
1	1	2	3	4	5	6	7	
2	1	2	3	4	5	6	7	
3	1	2	3	4	5	6	7	
4	1	2	3	4	5	6	7	
5	1	2	3	4	5	6	7	
6	1	2	3	4	5	6	7	
7	1	2	3	4	5	6	7	
8								28
9	Tömb bal átlója elemeinek Összege							

a tömb átló összegének számolása

6.1.3 Tömbváltozó elemei közötti Min és Max értékek keresése

Tömbváltozóban megkereshetőek e legnagyobb (Max) és legkisebb (Min) értékek, illetve rögzíthető a tömbben elfoglalt pozíciójuk is. A mintapélda rutin, egy dimenziós tömb cellából történő feltöltésével kezdődik. Ezt követően a Max és Min változónak „kötelezően" kezdő értéket kell adni, amely értékadásnál legcélszerűbb a vizsgált „tömb" első elemének értékét megadni, mert ez az elem biztosan a tömb elemei között szerepel, így a többi elem hozzá hasonlítható. Ezután egy For … Next ciklusban, ismételten If … Then parancsban vizsgálni lehet a „tömb aktuális elemének" > vagy < értékeit, s találat esetén tárolni kell az értéket és a találat helyét.

Program	**Megjegyzés**
Sub min_max()	
Dim intA() As Integer	dinamikustömb deklarációja
Dim intMax%, intMaxh%	
Dim intMin%, intMinh%	
Dim intT%, intI%	
Rows(2).ClearContents	célcellák (2. sor) előtörlése
intT = 0	
Do	
intT = intT + 1	
ReDim Preserve intA(intT)	intA() tömb méretének növelése
intA(intT) = Cells(1, intT)	intA() tömb feltöltése
Loop While IsEmpty(Cells(1,intT+1)) = False	
Cells(2, intT) = intT	
Cells(3, intT) = „Szám"	
intMax = intA(1): intMaxh = -32768	intMax és intMin kezdőérték megadása
intMin = intA(1): intMinh = 32767	az integer adattípus értékkészlete alapján
For intI = 1 To intT	intMax és intMin keresése
If intA(intI) < intMin Then	
intMin = intA(intI): intMinh = intI	
ElseIf intA(intI) > intMax Then	
intMax = intA(intI): intMaxh = intI	
End If	
Next intI	
Cells(2, intMaxh) = intMax	Találat kiírása, találati helyegyezéssel
Cells(3, intMaxh) = „Max"	
Cells(2, intMinh) = intMin	
Cells(3, intMinh) = „Min"	
End Sub	



	A	B	C	D	E	F	G	H	I	J	K
1	3	7	12	25	2	11	1	8	9	2	10
2				25			1				11
3				Max			Min				Szám

6.1.4 Sorfüggvények és Faktoriális számolás rutin eljárások

6.1.4.1 Sorfüggvény példa

Adott az $f(k)=3/(k^2+5k+6)$ függvény. Megírandó a „Sorfuggveny” VBA program, amely a $\text{Sum} = \sum_{k=1}^{m} f(k) = f(1) + f(2) + ... + f(m)$ összeget számítja ki és az összeget a felhasználó által meghatározott „p” értéktől ki is írja az Excel cellákba.

Program

```
Sub Sorfuggveny()

Dim intM%, intK%, intP%
Dim sngFk!, sngSum!

intM = InputBox("m = ?", "Adatbekeres", 5)
intP = InputBox("p = ?", "Adatbekeres", 3)

sngSum = 0
Cells(1, 1) = "k": Cells(1, 2) = "fk": Cells(1, 3) = "sum"

For intK = 1 To intM
   Cells(intK + 3, 1) = intK
   sngFk = 3 / (intK ^ 2 + 5 * intK + 6)
   Cells(intK + 3, 2) = Round(sngFk, 3)
   sngSum = sngSum + sngFk

   If intK >= intP Then
      Cells(intK + 3, 3) = Round(sngSum, 3)
   End If

Next intK


End Sub
```

Megjegyzés

Beolvasások InputBox paranccsal

	A	B	C
1	k	fk	sum
2			
3			
4	1	0,250	
5	2	0,150	
6	3	0,100	0,500
7	4	0,071	0,571
8	5	0,054	0,625

Három tizedesre kerekít és cellába ír.

Az összeg kiírásának feltételét vizsgálja.

6.1.4.2 Faktoriális számítása

A Faktorialis programot beolvassa x és n értékeket, majd kiszámolja a $1, x, \frac{x^2}{2!}, \frac{x^3}{3!}, \ldots \frac{x^n}{n!}$ faktoriális sorfüggvény tagjainak értékét és a tagok összegét. Megfigyelhető, hogy a tagok egymásból $\frac{x}{i}$ értékkel való szorzással állíthatóak elő, ahol az i 1-től n-ig levő egész számok halmaza.

Program

```
Sub faktorialis()

Dim intX%, intN%, sngTag!
Dim sngSum!, intI%

intN = InputBox("N= ?", "Beolvasás", 11)
intX = InputBox("X= ?", "Beolvasás", 3)

Cells(1, 1) = "N": Cells(1, 2) = "X"
Cells(1, 3) = "Tag"

sngTag = 1: Cells(2, 3) = sngTag
sngSum = 0

For intI = 1 To intN
   sngTag = sngTag * (intX / intI)
   Cells(intI + 2, 1) = intI
   Cells(intI + 2, 3) = sngTag
   sngSum = sngSum + sngTag                    tagok összegzése
Next intI

   Cells(intN + 3, 2) = "Sum"
   Cells(intN + 3, 3) = sngSum

End Sub
```

Megjegyzés

	A	B	C
1	N	X	Tag
2			1
3	1		3
4	2		4,5
5	3		4,5
6	4		3,375
7	5		2,025
8	6		1,0125
9	7		0,433929
10	8		0,162723
11	9		0,054241
12	10		0,016272
13	11		0,004438
14		Sum	19,0841

6.1.5 Műveletek vektorokkal és tömbökkel

6.1.5.1 Vektorok összege, szorzata, hossza

Összeg: $A+B=(A(1)+B(1)),(A(2)+B(2)),\ldots(A(N)+B(N))$

Szorzat: $A*B=(A(1)*B(1))+(A(2)*B(2))+\ldots(A(N)PB(N))$

Hossz: $A\ hossz=Sqr((A(1)*B(1))+(A(2)*B(2))+\ldots(A(N)PB(N)))$

Program

```
Sub vektor()

Dim intA%(3), intB%(3)
Dim intC%(3), intE%, intI%

For intI = 1 To 3
   intA(intI) = Cells(3, intI + 1)
   intB(intI) = Cells(5, intI + 1)
   intC(intI) = intA(intI) + intB(intI)
   Cells(7, intI + 1) = intC(intI)
Next intI

intE = 0

For intI = 1 To 3
   intE = intE + intA(intI) * intB(intI)
Next intI

Cells(9, 2) = intE
intE = 0

For intI = 1 To 3
   intE = intE + intA(intI) ^ 2
Next intI

Cells(11, 2) = Sqr(intE)
intE = 0

For intI = 1 To 3
   intE = intE + intB(intI) ^ 2
Next intI

Cells(13, 2) = Sqr(intE)

End Sub
```

Megjegyzés

A vektor beolvasása
B vektor beolvasása
A + B összeadás
A + B kiírása

A * B számolás

A hossz négyzetének számolása

B hossz négyzetének számolása

	A	B	C	D
1	Vektor			
2				
3	A	4	2	9
4				
5	B	2	-1	6
6				
7	A + B	6	1	15
8				
9	A * B	60		
10				
11	A hossz	10,04988		
12				
13	B hossz	6,403124		

6.1.5.2 Tömbök szorzása

A tömbök szorzásának feltétele, hogy egyik dimenziójuk nagysága azonos kell legyen. A mintapélda egy A(2 x 3) tömb szorzása B(3 x 4) tömbbel, melynek eredménye C(2 x 4) tömb. A tömbök elhelyezése az Excel lapon csak a „vizuális látvány” és az érthetőség segítését szolgálja.

Program **Megjegyzés**

```
Sub tömb_szorzas()

Dim intA%(2, 3), intB%(4, 3), intC%(3, 4)
Dim intI%, intK%, intJ%

For intI = 1 To 2
   For intK = 1 To 3
      intA(intI, intK) = Cells(intI + 1, intK)        „A” tömb feltöltése
   Next intK
Next intI

For intI = 1 To 4
   For intK = 1 To 3
      intB(intI, intK) = Cells(intK + 4, intI + 4)    „B” tömb feltöltése
   Next intK
Next intI

Range("E2:F3").ClearContents                          „C” tömb helyének törlése

For intI = 1 To 2                                     3x  For … Next számoló
   For intJ = 1 To 4
      s = 0

      For intK = 1 To 3
         s= s+(intA(intI, intK) * intB(intJ, intK))
      Next intK

      intC(intI, intJ) = s
      Cells(intI + 1, intJ + 4) = s
   Next intJ
Next intI

End Sub
```

	A	B	C	D	E	F	G	H
1	A tömb				A * B			
2	2	6	-1		29	36	-11	29
3	5	2	7		74	88	-4	43
4								
5					3	4	5	6
6					5	6	-4	3
7					7	8	-3	1
8								
9					B tömb			

6.1.6 Kisebb – nagyobb rendezés (Buborékrendezés)

Program

```
Sub novekvo_rendezes()

Dim intK%, intI%
Dim intA%, intB%, intC%

For intK = 1 To 10
   For intI = 1 To 10 - (intK - 1)
      intA = Cells(intI, 1)
      intB = Cells(intI + 1, 1)
      If intA > intB Then
         intC = intA
         Cells(intI, 1) = intB
         Cells(intI + 1, 1) = intC
      End If
   Next intI
Next intK

End Sub
```

Megjegyzés

A program futtatása előtt véletlenszerűen ki kell tölteni az A1:A10 cellákat az 1-10 számokkal.

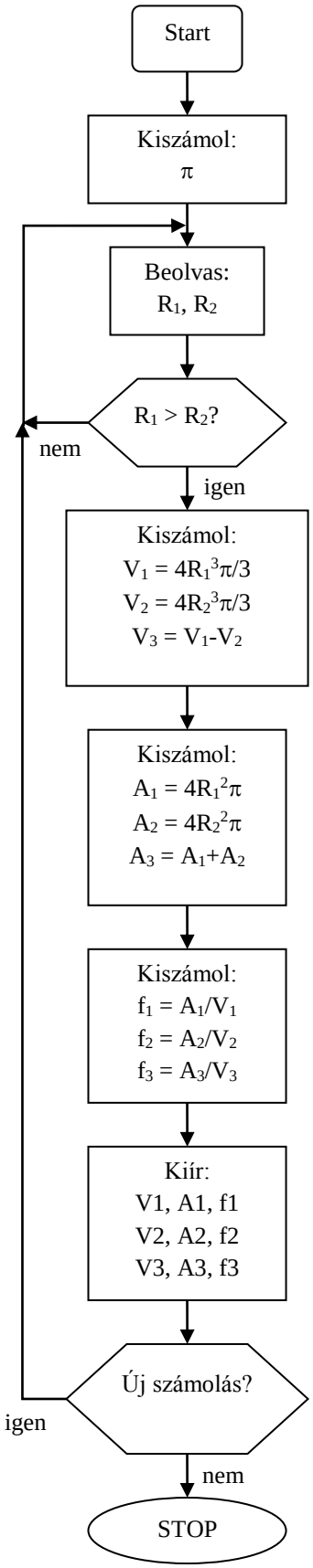
	A
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10

6.2 Kidolgozott gyakorló feladatok

6.2.1 Program írása blokkdiagram alapján

6.2.1.1 Térfogatszámolás feladat

Írja meg a „Terfogatszamolas” nevű VBA programot a mellékelt blokkdiagram alapján az if – then – goto alkalmazásával! Az adatbevitelhez az InputBox, üzenetkiíráshoz a MsgBox parancsot használja! Értelmezze a program működését! Indokolja meg, miért szükséges az R1 > R2 kitétel! A program megírása során az alábbi változókat alkalmazza: sngR1, sngR2, sngV1, sngV2, sngV3, sngA1, sngA2, sngA3, sngf1, sngf2, sngf3, sngPi (mind single) és strDontes (string). A ciklusok, tömbök, függvények és a szubrutinok alkalmazásának gyakorlása során térjen vissza a feladatra és egyszerűsítse a programot!



Megoldás:

Először a programnak kell nevet adni, majd ezt követi a változók deklarálása.

```
Sub Terfogatszamolas()

Dim sngR1 As Single, sngR2!
Dim sngV1 As Single, sngV2!, sngV3!
Dim sngA1 As Single, sngA2!, sngA3!
Dim sngf1 As Single, sngf2!, sngf3!
Dim strDontes As String
Dim sngPi As Single
```

A következő lépésben π értékét kell kiszámolni a matematikai függvényeknél leírtak alapján.

```
sngPi = 4 * Atn(1)
```

A blokkdiagramon látszik, hogy mielőtt R_1 és R_2 értékét beolvasná a program, meg kell oldani a program végéről a visszavezetést, melyre az „ujra” címke szolgál:

```
ujra:
sngR1 = InputBox("R1 = ?", "Adatbekeres", 2)
sngR2 = InputBox("R2 = ? (kisebb, mint R1!)", "Adatbekeres", 1)
```

Ezek után le kell ellenőrizni, hogy $R_1>R_2$, amennyiben a feltétel nem teljesül, R_1 és R_2 értékét értékét újra be kell olvasni, amihez szintén az „ujra” címkét kell felhasználni:

```
If sngR2 >= sngR1 Then GoTo ujra
```

Ezt követi a térfogatok (V) és a felületek (A) és a fajlagos felületek (f) kiszámolása:

```
sngV1 = 4 * sngR1 ^ 3 * sngPi / 3
sngV2 = 4 * sngR2 ^ 3 * sngPi / 3
sngV3 = sngV1 - sngV2
sngA1 = 4 * sngR1 ^ 2 * sngPi
sngA2 = 4 * sngR2 ^ 2 * sngPi
sngA3 = sngA1 + sngA2
sngf1 = sngA1 / sngV1
sngf2 = sngA2 / sngV2
sngf3 = sngA3 / sngV3
```

Az „1” és „2” indexű változók egy-egy gömb térfogatára, felületére és fajlagos felületére vonatkoznak, míg a „3” indexű változók a két gömb tulajdonságainak különbségére, azaz a nagyobbik gömb „héjára” (például narancshéj) vonatkoznak. A térfogat csak pozitív szám lehet, ezért szükséges, hogy $R_1>R_2$ legyen. Ezután a MsgBox paranccsal ki kell írni a számolt értékeket. A szöveg és szám típusú változók összefűzésére a „&” karakter szolgál. Nem szabad elfelejteni, hogy a kiírandó szöveget idézőjelek közé kell tenni!

```
MsgBox "V1 = " & sngV1 & ", A1 = " & sngA1 & ", f1 = " & sngf1, , "1. adatainak kiirasa"
MsgBox "V2 = " & sngV2 & ", A2 = " & sngA2 & ", f2 = " & sngf2, , "2. adatainak kiirasa"
MsgBox "V3 = " & sngV3 & ", A3 = " & sngA3 & ", f3 = " & sngf3, , "3. adatainak kiirasa"
```

Végezetül meg kell kérdezni, hogy a felhasználó kíván új számolást végezni vagy nem. Igen válasz esetén vissza kell térni az „ujra” címkéhez, míg nem válasz esetén a program véget ér.

```
strDontes = InputBox("Uj szamolas? (i = igen, bármi más = nem)", "Adatbekeres", "i")
If strDontes = "i" Then GoTo ujra

MsgBox "Program vege"

End Sub
```

6.2.2 Program írása blokkdiagram nélkül

6.2.2.1 Öröknaptár feladat

Írja meg az „Oroknaptar” nevű programot, amely megadja, hogy egy adott dátum a hét melyik napjára esett/fog esni. Referenciadátum: 2009. 08. 31., hétfő. A megoldás során használja a maradékosztásra szolgáló „mod” parancsot, a „Select Case” utasítást, valamint az alábbi változókat: datReferencia, datKerdesesNap (mindkettő date), sngKulonbseg (single, a két dátum különbsége), intMaradek (integer, maradékosztás eredménye). Ne feledkezzen el arról, hogy sngKulonbseg és intMaradek értéke negatív is lehet! Miért nem használhatunk az sngKulonbseg változó helyett integer típusú intKulonbseg változót? Az intMaradek változó helyett alkalmazhatnánk byte típusú bytMaradek változót? A tömbökről tanultak után térjen vissza a feladatra, írja át a megoldást!

Megoldás:

Először a változókat kell deklarálni:

```
Sub Oroknaptar()

Dim datReferencia As Date, datKerdesesNap As Date
Dim intMaradek As Integer
Dim sngKulonbseg As Single
```

Ezt követi a referenciadátum megadása, illetve a kérdéses nap dátumának bekérése a felhasználótól:

```
datReferencia = #8/31/2009#
datKerdesesNap = InputBox("Kerem, adja meg a kerdeses nap datumat!", "Adatbekeres", #1/1/2009#)
```

A dátumokat a Visual Basic számokként kezeli, így különbségük is képezhető. A különbség lehet pozitív, illetve negatív szám is, mert a kérdéses nap dátuma lehet a referencianapnál későbbi, illetve korábbi dátum is. A különbséget el kell osztani héttel és meg kell határozni a maradékot, mely egyértelműen megadja, hogy a kérdéses dátum milyen napra esett.

```
sngKulonbseg = datKerdesesNap - datReferencia
intMaradek = sngKulonbseg Mod 7

Select Case intMaradek
   Case 0
      MsgBox "A kerdeses nap hetfo volt/lesz."
   Case -6, 1
      MsgBox "A kerdeses nap kedd volt/lesz."
   Case -5, 2
      MsgBox "A kerdeses nap szerda volt/lesz."
   Case -4, 3
      MsgBox "A kerdeses nap csutortok volt/lesz."
```

```
    Case -3, 4
        MsgBox "A kerdeses nap pentek volt/lesz."
    Case -2, 5
        MsgBox "A kerdeses nap szombat volt/lesz."
    Case -1, 6
        MsgBox "A kerdeses nap vasarnap volt/lesz."
End Select

End Sub
```

6.2.3 Gyakorlatok For - Next és Do - Loop ciklusok alkalmazására

6.2.3.1 Lottó feladat

Írjon VBA programot a For-Next és a Do-Loop (elől és hátul tesztelő) ciklus alkalmazásával, amelynek segítségével kiszámítható a nyerési esély a Magyarországon forgalmazott lottók (5/90, 6/45) esetében. Az adatok beolvasásához az InputBox (a lottószelvényen lévő számok és a kihúzandó számok mennyisége, bytN és bytK, mindkettő byte típusú változó), az adatok kiírásához a MsgBox parancsot használja. A nyerési esély a $\frac{n!}{(n-k)!k!}$ képlet segítségével számítható ki. A gyorsabb számolás érdekében a fenti képlet alábbi egyszerűsített változatát használja: $\frac{n(n-1)...(n-k+1)}{k!} = \prod_j \frac{n-j}{1+j}$; ahol $j = 0,1,...,k-1$. A programok írásakor használjon modulszintű változókat.

Megoldás:

A modulszintű változókat célszerű az „Option” parancsok alatt, de az első program nyitó „Sub” sora felett deklarálni:

```
Option Explicit

Dim bytJ As Byte, bytK As Byte, bytN As Byte
Dim sngNyeresiEsely As Single

Sub Lotto_For()
```

A sngNyeresiEsely változó tartalmazza majd az összes lehetséges húzásvariáció számát, míg a bytJ változó a For - Next ciklus ciklusváltozója. A deklarációt követi a kihúzható lottószámok, valamint a húzások számának megadása:

```
bytN = InputBox("Kihuzhato lottoszamok szama (max. 255)", "Adatbekeres", 90)
bytK = InputBox("Huzasok szama (max. 255)", "Adatbekeres", 5)
```

A következő lépésben a sngNyeresiEsely változónak kiindulási értéket kell adni, csak ezután következhet

a For - Next ciklus írása. A képlet szerint a For - Next ciklusban a törteket szorozni kell egymással, így sngNyeresiEsely változó kiindulási értéke 1:

```
sngNyeresiEsely = 1

For bytJ = 0 To bytK - 1
   sngNyeresiEsely = sngNyeresiEsely * (bytN - bytJ) / (1 + bytJ)
Next bytJ
```

Végezetül a kapott eredményt kell kiírni:

```
MsgBox "Az adott lottonal a nyeresi esely = 1 : " & sngNyeresiEsely & ".", , "Eredmeny"

End Sub
```

Természetesen a feladat Do Loop ciklussal is megoldható, ebben az esetben a

```
For bytJ = 0 To bytK - 1
   sngNyeresiEsely = sngNyeresiEsely * (bytN - bytJ) / (1 + bytJ)
Next bytJ
```

programrészlet helyett elől tesztelő Do - Loop ciklust alkalmazva a

```
bytJ = 0

Do While bytJ <= bytK - 1
   sngNyeresiEsely = sngNyeresiEsely * (bytN - bytJ) / (1 + bytJ)
   bytJ = bytJ + 1
Loop
```

programrészletet, míg hátul tesztelő Do - Loop ciklust alkalmazva a

```
bytJ = 0

Do
   sngNyeresiEsely = sngNyeresiEsely * (bytN - bytJ) / (1 + bytJ)
   bytJ = bytJ + 1
Loop While bytJ <= bytK - 1
```

programrészletet kell használni. Látható, hogy a For - Next ciklushoz képest a Do - Loop ciklus használatakor a ciklus előtt bytJ változónak kiindulási értéket kell adni, illetve a ciklus futása során bytJ értékét meg kell növelni.

6.2.3.2 Pithagorasz-féle számhármasok meghatározása

Írjon VBA programot Do-Loop (előltesztelő)és a For-Next ciklus alkalmazásával. amely meghatározza az összes olyan Pithagorasz-féle számhármast, amelyben a három egész szám egyike sem nagyobb, mint 100. A három szám (intA, intB és intC (integer típusú változók)) értékeit külön oszlopokban, a 2. sortól kezdve írassa ki egymás alatti cellákba (Az első sor fejlécként az „A”, „B” és „C” betűket tartalmazza.). A programok írásakor használjon modulszintű változókat.

Megoldás:

A program írása előtt értelmezni kell a feladat szövegét. A derékszögű háromszög leghosszabb oldala az átfogó (intC), így a „három egész szám egyike sem nagyobb, mint 100” kitétel gyakorlatilag azt jelenti, hogy intC ≤ 100. A két befogó (intA, intB) közül az egyik befogó (intB) hosszának nagyobbnak kell lennie, mert intA = intB esetén az $intA^2 + intB^2 = intC^2$ összefüggés értelmében $intC^2 = 2 \times intB^2$, amiből egyértelmű, hogy intC és intB egyszerre nem lehet egész szám. A fentebb leírtak alapján intA < intB < intC. A feladat legegyszerűbben három, egymásba ágyazott ciklus segítségével oldható meg, ahol intC értéke a 3 - 100, intB értéke a 2 - intC-1, míg intA értéke a 1 - intB-1 tartományban változhat.

```
Sub Pithagorasz_For()

Dim intA As Integer, intB%, intC%, intI%
```

intI változóra intA, intB és intC egymás alatti sorokban történő kiírása miatt van szükség.

```
Cells(1, 1) = "A"
Cells(1, 2) = "B"
Cells(1, 3) = "C"
```

Az A1:A3 cellákba kerül a kiírandó adatok táblázatának fejléce. A következő lépésben intI változónak kell kiindulási értéket adni. Az első Pithagorasz-féle számhármast a B1:B3 cellákba (azaz a második sorba) kell kiírni, ezután következhet az egymásba ágyazott For-Next ciklusokmegnyitása:

```
intI = 2

For intC = 3 To 100
   For intB = 2 To intC - 1
      For intA = 1 To intB - 1
```

A következő lépésben meg kell vizsgálni, hogy teljesül-e az $intA^2 + intB^2 = intC^2$ feltétel. Amennyiben a feltétel teljesül, akkor a változók értékeit ki kell írni az Excel munkalapra, illetve intI értékét meg kell növelni, hogy a következő számhármas kiírása ne ugyanabba a sorba történjen. Ezt követi a ciklusok zárása:

```
            If intC ^ 2 = intA ^ 2 + intB ^ 2 Then
               Cells(intI, 1) = intA
               Cells(intI, 2) = intB
               Cells(intI, 3) = intC
               intI = intI + 1
            End If

        Next intA
    Next intB
Next intC

End Sub
```

Látható, hogy egymásba ágyazott ciklusok esetén először a legbelső ciklust kell bezárni, míg utolsónak a legkülső ciklus bezárása marad.

A programon lehet finomítani, hiszen a harmadik ciklusra tulajdonképpen nincs szükség, mert a Pithagorasz tétel alapján két oldal hosszának ismeretében a harmadik oldal hossza számítható. Ugyanakkor az így számított érték a legtöbbször nem egész szám, tehát a programozás során egy új, single típusú változót (sngA) is deklarálni kell:

```
Sub Pithagorasz_For_2()

Dim intB%, intC%, intI%
Dim sngA as Single

Cells(1, 1) = "A"
Cells(1, 2) = "B"
Cells(1, 3) = "C"

intI = 2

For intC = 3 To 100
    For intB = 2 To intC - 1
```

Eddig a program megegyezik az előző programmal, de most kerül sor sngA értékének kiszámítására és annak vizsgálatára, hogy sngA egész szám vagy sem:

```
        sngA = Sqr(intC ^ 2 - intB ^ 2)
        If sngA = Int(sngA) And intB > Int(sngA) Then
            Cells(intI, 1) = sngA
            Cells(intI, 2) = intB
            Cells(intI, 3) = intC
            intI = intI + 1
        End If
    Next intB
Next intC

End Sub
```

Az Sqr függvény a gyökvonást végzi. Természetesen jó megoldás a

```
sngA = (intC ^ 2 - intB ^ 2) ^ 0.5
```

parancs is. Az Int(sngA) kifejezés sngA értékének egész részét állítja elő, amely természetesen csak akkor lehet egyenlő sngA értékével, ha sngA egész szám. Az intB > Int(sngA) feltételre nem csak azért van szükség, mert korábban kikötöttük, hogy intB > intA, hanem azért is, hogy a program ugyanazt a megoldást ne kétszer találja meg (intA = 3, intB = 4, intC = 5; illetve intA = 4, intB = 3, intC = 5).
A feladat természetesen megoldható Do - Loop ciklus alkalmazásával is:

```
Sub Pithagorasz_Do()

Dim intB%, intC%, intI%
Dim sngA as Single

Cells(1, 9) = "A"
Cells(1, 10) = "B"
Cells(1, 11) = "C"

intC = 3
intI = 2

Do While intC <= 100
   intB = 2
   Do While intB < intC
      sngA = Sqr(intC ^ 2 - intB ^ 2)
      If sngA = Int(sngA) And intB > Int(sngA) Then
         Cells(intI, 9) = sngA
         Cells(intI, 10) = intB
         Cells(intI, 11) = intC
         intI = intI + 1
      End If
      intB = intB + 1
   Loop
   intC = intC + 1
Loop

End Sub
```

Látható, hogy a Pithagorasz_Do program csak annyiban Pithagorasz_For_2 programtól, hogy előbbiben a Do - Loop ciklusok alkalmazása előtt intC és intB változóknak kiindulási értéket kell adni (3, illetve 2), valamint intC és intB változók értékét a ciklusuk vége előtt meg kell növelni.

6.2.3.3 Számkitaláló feladat

Írjon olyan VBA programot Do-Loop (hátul tesztelő) ciklus alkalmazásával, amelyben a számítógép kigondol egy 1 és 100 közé eső egész számot (intGondol, integer), majd felszólítja a felhasználót, hogy találja ki azt (MsgBox utasítás). A felhasználó tippjére (intTipp, integer; InputBox utasítás) közli, hogy az nagyobb, vagy kisebb az általa gondoltnál, illetve a helyes tipp esetén gratulál és felajánlja, hogy új játékot kezdjenek (strDontes, string). Kérdés: A feladat megoldható For-Next ciklus alkalmazásával?

Megoldás:

A program írása előtt végig kell gondolni, hány ciklus alkalmazására van szükség. Ciklust kell alkalmazni akkor, amikor

- a program megkérdezi, hogy a felhasználó kíván-e új játékot játszani,
- a program bekéri a felhasználó következő tippjét.

A két ciklus közül egyértelműen a tipp bekérése a belső ciklus, hiszen a tippkérés a játékon **belül** történik, míg az új játékra vonatkozó ciklus a külső ciklus, mert az új játékra vonatkozó kérdés a játékon **kívül**, két játék között történik.

A program írása során először deklarálni kell a használt változókat:

```
Sub Kitalalo()

Dim intGondol As Integer, intTipp%
Dim strDontes As String
```

Ezt követi a külső ciklus megnyitása és a véletlen szám generálása:

```
Do
   intGondol = Int(100 * Rnd(1)) + 1
   MsgBox "Kerem, talalja ki melyik szamra gondoltam!"
```

A Rnd függvény a 0-1 tartományban generál számot, amit a feladat „1 és 100 közé eső egész szám” kitétele miatt előbb be kell szorozni százzal, majd az így kapott szám egész részét (0, 1, 2, … 99) kell venni az Int függvénnyel, végül meg kell az értékét növelni eggyel. Most kerülhet sorra a belső ciklus megnyitása majd a tipp bekérése:

```
   Do
      intTipp = InputBox("Melyik szamra tippel?", "Adatbekeres")
```

A beadott tippet össze kell hasonlítani a számítógép által generált számával, majd közölni kell a felhasználóval, hogy a két érték hogyan viszonyul egymáshoz:

```
Select Case intTipp
   Case Is < intGondol
      MsgBox "Nagyobb szamra gondoltam!"
   Case Is > intGondol
      MsgBox "Kisebb szamra gondoltam!"
   Case intGondol
      MsgBox "Gratulalok! Kitalalta a szamot!"
End Select
```

A Select Case - End Select szerkezet helyett az if - else - End If szerkezet is használható, ebben az esetben a fenti programrészlet helyett az alábbi programrészletet kell megírni:

```
If intTipp < intGondol Then
   MsgBox "Nagyobb szamra gondoltam!"
   ElseIf intTipp > intGondol Then
      MsgBox "Kisebb szamra gondoltam!"
      Else
         MsgBox "Gratulalok! Kitalalta a szamot!"
End If
```

Látható, hogy ebben az esetben már nincs értelme az Else után újabb feltételt megadni (bár lehet), hiszen, ha intTipp se nem nagyobb, se nem kisebb, mint intGondol, akkor csak egyenlő lehet vele. Most kerülhet sor a belső ciklus bezárására:

```
   Loop Until intGondol = intTipp
```

Természetesen a

```
   Loop While intGondol <> intTipp
```

parancs is megfelel. Mivel a belső ciklus bezárásával a játék is véget ért, a felhasználót meg kell kérdezni, hogy kíván új játékot kezdeni vagy sem. Ezt követi a külső ciklus bezárása és a program vége:

```
   strDontes = InputBox("Van kedve uj jatekot kezdeni? (i = igen, barmi mas = nem)", "Adatbekeres", "i")
Loop While strDontes = "i"

End Sub
```

Természetesen a külső ciklus bezárása a

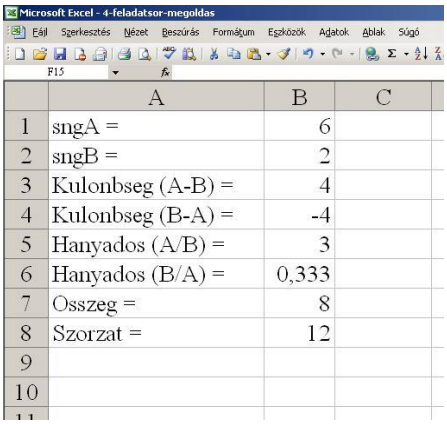
```
Loop Until strDontes <> "i"
```

paranccsal is megtörténhet.

6.2.4 Gyakorlatok Function függvények és szubrutinok alkalmazására

6.2.4.1 Alapműveletek feladat

Írja meg az „Alapmuveletek” VBA programot, amely függvények (Function) felhasználásával kiszámítja két tetszőleges szám (sngA és sngB, pl: 6 és 2) összegét, különbségét, szorzatát és hányadosát. A kapott eredményeket a jobboldali ábrának megfelelően írassa ki.



Utána írja meg az „Alapmuveletek2” VBA programot, amely ugyanazt végzi, mint az „Alapmuveletek”, csak függvények helyett szubrutinokat használ. A programok írásakor használjon modulszintű változókat. (sngA, sngB, sngKulonbseg1, sngKulonbseg2, sngHanyados1 és sngHanyados2 single típusú változók, dblOsszeg és dblSzorzat double típusú változók).

Megoldás:

Első lépésben deklaráljuk a programok írása során használt változókat. Mivel mindkét program ugyanazokat a változókat használja, célszerű modulszintű változókat használni.

```
Option Explicit

Dim sngA As Single, sngB!, sngKulonbseg1!, sngKulonbseg2!
Dim sngHanyados1!, sngHanyados2!
Dim dblOsszeg As Double, dblSzorzat#

Sub Alapmuveletek()
```

A deklarációt sngA és sngB változók beolvasása, majd munkalapra történő írása követheti:

```
sngA = InputBox("Kerem A erteket!", "Adatbekeres", 6)
sngB = InputBox("Kerem B erteket!", "Adatbekeres", 2)

Cells(1, 1) = "sngA =": Cells(1, 2) = sngA
Cells(2, 1) = "sngB =": Cells(2, 2) = sngB
```

A megadott két szám értékének ismeretben meghívhatóak a különbség, összeg, hányados és szorzat értékeket számítható függvények. A függvényekben közös, hogy mindegyiknek át kell adni sngA és sngB értékét, míg a függvények az eredményt a nevükkel adják vissza:

```
sngKulonbseg1 = Kivon(sngA, sngB): sngKulonbseg2 = Kivon(sngB, sngA)
sngHanyados1 = Eloszt(sngA, sngB): sngHanyados2 = Eloszt(sngB, sngA)
dblOsszeg = Osszead(sngA, sngB)
dblSzorzat = Osszeszoroz(sngA, sngB)
```

A „Kivon” függvény második hívása helyett alkalmazható a

```
sngKulonbseg2 = -sngKulonbseg1
```

utasítás is. A kiszámolt értékek kiírhatóak az Excel munkalapra:

```
Cells(3, 1) = "Kulonbseg (A-B) =": Cells(3, 2) = sngKulonbseg1
Cells(4, 1) = "Kulonbseg (B-A) =": Cells(4, 2) = sngKulonbseg2
Cells(5, 1) = "Hanyados (A/B) ="
```

A hányados számolásánál gond lehet, amennyiben a nevezőben lévő szám értéke 0. Az $sngA \neq 0$, $sngB \neq 0$ feltételek vizsgálata történhet az „Osztas” függvény hívása előtt, vagy a függvényben. Ha a programban a függvényt többször hívják meg, célszerű lehet a vizsgálatot a függvényben elvégezni. Ez az oka, hogy a

```
sngHanyados2 = 1/ sngHanyados1
```

parancsok nem minden esetben jó megoldások. A hányadosok értékének munkalapra történő kiírásakor viszont jelezni kell a felhasználónak, ha a hányados nem számítható:

```
If sngB = 0 Then
   Cells(5, 2) = "Nem ertelmezheto"
   Else
   Cells(5, 2) = sngHanyados1
End If

Cells(6, 1) = "Hanyados (B/A) ="

If sngA = 0 Then
Cells(6, 2) = "Nem ertelmezheto"
Else
Cells(6, 2) = sngHanyados2
End If
```

Végezetül az Excel munkalapra a két szám összegének és szorzatának értékét kell kiírni:

```
Cells(7, 1) = "Osszeg =": Cells(7, 2) = dblOsszeg
Cells(8, 1) = "Szorzat =": Cells(8, 2) = dblSzorzat

End Sub
```

A főprogram megírása után kerülhet sor a függvények megírására. Noha mindegyik függvény sngAf és sngBf változókat használ, de ez nem okoz hibát, mert mindegyik változó hatóköre csak a saját függvényére terjed ki.

```
Function Kivon(sngAf!, sngBf!) As Single
Kivon = sngAf - sngBf
End Function

Function Osszead(sngAf!, sngBf!) As Double
Osszead = sngAf + sngBf
End Function

Function Osszeszoroz(sngAf!, sngBf!) As Double
Osszeszoroz = sngAf * sngBf
End Function

Function Eloszt(sngAf!, sngBf!) As Single
If sngBf = 0 Then
   MsgBox "Hiba! Oszto nem lehet 0!"
   Exit Function
   else
   Eloszt = sngAf / sngBf
End If
End Function
```

Látható, hogy a sngBf = 0 esetben az Eloszt függvény anélkül fejeződik be (Exit Function), hogy értéket a főprogram sngHanyados1 vagy sngHanyados2 változójának.

Az Alapműveletek2 program írásakor az Alapműveletek program jelentős része átvehető, eltérés csak abban van, hogy az Alapműveletek2 program függvények helyett szubrutinokat használ, melyeket a Call paranccsal kell meghívni:

```
Call Kivonasok(sngA, sngB, sngKulonbseg1, sngKulonbseg2)
Call Osztasok(sngA, sngB, sngHanyados1, sngHanyados2)
Call Osszeadas(sngA, sngB, dblOsszeg)
Call Osszeszorzas(sngA, sngB, dblSzorzat)
```

Végezetül a szubrutinokat kell megírni.

```
Sub Kivonasok(sngAf!, sngBf!, sngEredmeny1!, sngEredmeny2!)
sngEredmeny1 = sngAf - sngBf
sngEredmeny2 = sngBf - sngAf
End Sub

Sub Osszeadas(sngAf!, sngBf!, dblEredmeny1!)
dblEredmeny1 = sngAf + sngBf
End Sub

Sub Osszeszorzas(sngAf!, sngBf!, dblEredmeny1#)
dblEredmeny1 = sngAf * sngBf
End Sub

Sub Osztasok(sngAf!, sngBf!, sngEredmeny1!, sngEredmeny2!)
If sngBf = 0 Or sngAf = 0 Then MsgBox "Hiba! Oszto nem lehet 0!": Exit Sub
If sngBf <> 0 Then sngEredmeny1 = sngAf / sngBf
If sngAf <> 0 Then sngEredmeny2 = sngBf / sngAf
End Sub
```

Az Osztasok szubrutinban az sngAf $\neq 0$ és sngBf $\neq 0$ feltételek vizsgálatát is el kell végezni. A függvény Exit Function parancsához hasonlóan ebben az esetben az Exit Sub utasítás szolgál a főprogramba történő visszatérésre.

6.2.4.2 Reakció feladat

Írja meg a „Reakcio1” nevű programot, amely kiszámolja az „A → B” reakció lejátszódása alatt az „A” anyag koncentrációját [A].

A számolást kétféleképpen végezzük el, egyrészt a $\Delta[A] = -k \cdot [A] \cdot \Delta t$ (Euler féle) közelítéssel, mely szerint az „A” anyag koncentrációjának Δt időintervallum alatti változása arányos az „A” anyagnak a Δt időintervallum elején mért koncentrációjával. Az összefüggésben az arányossági tényező, a „k” reakciósebességi állandó értéke legyen 0,005 1/s)

A másik számolás során a pontos, $[A]_t = [A]_0 \cdot e^{-k \cdot t}$ összefüggést használjuk, ahol [A]0 az „A” anyag t = 0 időpontban mért koncentrációja, [A]t az „A” anyagnak a reakció elindítása utáni „t” időpontban mért koncentrációja.

[A] értékét az 1. módszer esetén a Δ[A] koncentrációváltozást számoló EULER nevű, Double típusú Function hívásával számolja , melynek bemenő paraméterei: A ([A] az időintervallum elején), k és dt (Δt).

Számítsa ki az Euler féle közelítés relatív hibáját is:

$$\text{Hiba}\,(\%) = 100\% \times \frac{\left|[A]_{\text{Euler}} - [A]_{\text{exp}}\right|}{[A]_{\text{exp}}},$$

ahol $[A]_{\text{Euler}}$ az Euler közelítéssel, $[A]_{\text{exp}}$ az exponenciális kifejezéssel számolt érték. A program az InputBox parancscsal az alábbi értékeket kérje be: k (reakciósebességi állandó), tmax (a vizsgált időtartomány), n (az időintervallumok száma a vizsgált időtartományban) A0 (az A anyag kezdeti koncentrációja.

	A	B	C	D
1	k=	0,005	1/s	
2	Idő	A=A-k*A*dt	A=A0*exp(-k*t)	Hiba
3	s	M	M	%
4	0	0,2	0,2	0
5	20	0,1800000	0,1809675	0,535
6	40	0,1620000	0,1637462	1,066
7	60	0,1458000	0,1481636	1,595
8	80	0,1312200	0,1340640	2,121
9	100	0,1180980	0,1213061	2,645
10	120	0,1062882	0,1097623	3,165
11	140	0,0956594	0,0993171	3,683
12	160	0,0860934	0,0898658	4,198
13	180	0,0774841	0,0813139	4,710
14	200	0,0697357	0,0735759	5,219
15	220	0,0627621	0,0665742	5,726
16	240	0,0564859	0,0602388	6,230
17	260	0,0508373	0,0545064	6,731
18	280	0,0457536	0,0493194	7,230
19	300	0,0411782	0,0446260	7,726

A programot futtassa az „F2n15” munkalapon k=0,005, tmax=300, n=15, A0=0,2 bemenő adatokkal és az „F2n300” munkalapon való futtatáshoz k, tmax és A0 bemenő adatok legyenek az előző futtatás megfelelő adataival egyenlőek, n értéke pedig legyen 300. Hasonlítsa össze az eredményeket!

Megoldás:

A változók deklarálása után az InputBox parancs alkalmazásával a felhasználótól be kell kérni a reakciósebesség, a vizsgált időtartomány, az időintervallumok számának, valamint az anyag kezdeti koncentrációjának értékét és ki kell tölteni a táblázat fejlécét. A vizsgált időtartomány és az időintervallumok számának ismeretében az időintervallum számítható.

```
Sub Reakcio1()

Dim i%, n%, k#, tmax#, dt#, t#, A0#, Akoz#, Apont#, hiba#

k = InputBox("k=?", "Adatbekeres", 0.005)
Cells(1, 1) = "k=": Cells(1, 2) = k: Cells(1, 3) = "1/s"
tmax = InputBox("tmax?", , 300): n = InputBox("n?", , 15)
dt = tmax / n
A0 = InputBox("A0?", , 0.2)
Cells(2, 1) = "Idő": Cells(3, 1) = "s"
Cells(2, 2) = "A=A-k*A*dt": Cells(3, 2) = "M"
Cells(2, 3) = "A=A0*exp(-k*t)": Cells(3, 3) = "M"
Cells(2, 4) = "Hiba": Cells(3, 4) = "%"
```

A következő lépésben megadható a t = 0 időpillanathoz tartozó kezdeti koncentráció értéke. Bár az alábbiakban az Apont értékét az Expo1 függvény segítségével számolja ki a program, de az Apont = 0.2 paranccsal is megadható. A hiba természetesen 0%. A táblázatba kiírandó értékeket a későbbiekben alkalmazandó i változó segítségével írja ki a program, de a Cells(4, 1) = t, Cells(4, 2) = Akoz, … parancsok is alkalmazhatóak.

```
i = 4: t = 0: Akoz = A0
Apont = Expo1(A0, k, t): hiba = 100 * Abs(Akoz - Apont) / Apont
Cells(i, 1) = t: Cells(i, 2) = Akoz
Cells(i, 3) = Apont: Cells(i, 4) = hiba
```

Az egyes időintervallumok utáni Akoz és Apont koncentráciértékek For - Next ciklus alkalmazásával és az Euler és Expo1 függvények segítségével számolhatóak. Az Euler függvény egyik bemenő értéke az aktuális Akoz értéke, melyre a $\Delta[A] = -k \cdot [A] \cdot \Delta t$ összefüggés miatt van szükség. A ciklusban i változó értékét meg kell növelni, hogy a számolt értékek kiírása a következő sorba történjen.

```
For t = dt To tmax Step dt
   Akoz = Euler(Akoz, k, dt)
   Apont = Expo1(A0, k, t)
   hiba = 100 * Abs(Akoz - Apont) / Apont
   i = i + 1
   Cells(i, 1) = t: Cells(i, 2) = Akoz
   Cells(i, 3) = Apont: Cells(i, 4) = hiba
Next t

End Sub
```

Végezetül az Euler és Expo1 függvényeket kell megírni. Az Euler függvény az első lépésben a koncentrációváltozás, a második lépésben az új koncentráció értékét számítja ki.

```
Function Euler(A#, k#, dt#) As Double
   Dim dA#
   dA = -A * k * dt
   Euler = A + dA
End Function
```

```
Function Expo1(A0#, k#, t#) As Double
    Expo1 = A0 * Exp(-k * t)
End Function
```

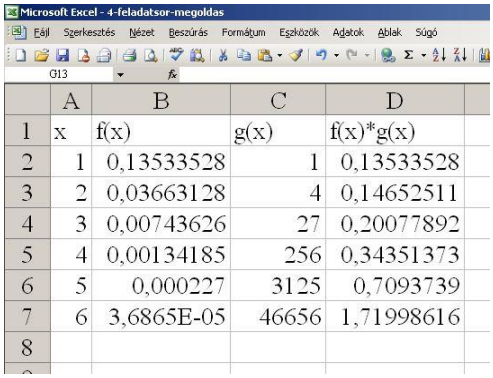
A programot n = 300 értékkel lefuttatva a tmax = 300 időpontban a hiba értéke csak 0,3755%, melynek oka, hogy az egyes időintervallumok huszadakkorák, mint n = 15 esetben, így az Euler féle közelítés közelebb van a valósághoz.

6.2.5 Fájlműveletek gyakorlatai

6.2.5.1 Kétfüggvényes feladat

Írja meg a „Szamolas” VBA programot, amely az Excel munkalap első sorába beírja a táblázat fejlécét, az „A” oszlopba Do-Loop ciklus alkalmazásával (ciklusváltozó: intI) beolvassa a „4-be.txt” fájl adatait (1, 2, 3, 4, 5, 6) az intBe változóba, majd függvények (Function) alkalmazásával kiszámítja a

$$f(x) = x * e^{-2x} \quad \text{és} \quad g(x) = x^x$$



	A	B	C	D
1	x	f(x)	g(x)	f(x)*g(x)
2	1	0,13533528	1	0,13533528
3	2	0,03663128	4	0,14652511
4	3	0,00743626	27	0,20077892
5	4	0,00134185	256	0,34351373
6	5	0,000227	3125	0,7093739
7	6	3,6865E-05	46656	1,71998616
8				

függvények értékét („B” és „C” oszlop). Utána a „B” és „C” oszlop megfelelő elemeit összeszorozva töltse ki „D” oszlop celláit. Végezetül For-Next ciklusok alkalmazásával (ciklusváltozó: intI, intJ) az összes adatot írja ki a „4-ki.txt” file-ba. (intI, intJ intBe és intVeg is integer típusú változó) Az f(x) és g(x) értékét számoló függvények milyen típusúak legyenek?

Megoldás:

A változók deklarálása után elkészíthető a táblázat fejléce, majd az intI változónak kell értéket adni. Az intI változóra a Do - Loop cikluson belül a f(x), g(x) és f(x) * g(x) értékek kiírásakor mint sorváltozóra lesz szükség. Mivel a cikluson belül intI változó értéke eggyel fog növekedni és az első értékeket a második sorba kell kiírni, így intI = 1 értéket célszerű adni.

```
Sub Szamolas()
Dim intI As Integer, intJ%, intBe%, intVeg%
Cells(1, 1) = "x": Cells(1, 2) = "f(x)": Cells(1, 3) = "g(x)": Cells(1, 4) = "f(x)*g(x)"
intI = 1
```

A következő lépésben a 4-be.txt fájlt kell megnyitni olvasásra és a Do - Loop ciklus segítségével kell be kell olvasni a fájl tartalmát. Javasolt elől tesztelő ciklust alkalmazni, mert „üres” 4-be.txt fájl esetében nincs beolvasandó adat és hátul tesztelő ciklust használva hibaüzenet jelenik meg. Általában nem ismert, hogy a fájlban mennyi adat van, ezért a beolvasást a file végéig (EOF: end of file, 1 a fájlazonosító) kell végezni:

```
Open "4-be.txt" For Input As #1

Do While Not EOF(1)
   intI = intI + 1
   Input #1, intBe
   Cells(intI, 1) = intBe
   intVeg = intI
Loop

Close #1
```

Az intVeg = intI utasítás biztosítja, hogy program elraktározza az utolsó, még adatokat tartalmazó sor számát. A következő lépésben az Exponencialis és Hatvany függvények segítségével és az intI ciklusváltozó értékét változtatva kiszámolhatóak f(x), g(x) és f(x) * g(x) értékek:

```
For intI = 2 To intVeg
   Cells(intI, 2) = Exponencialis(Cells(intI, 1))
   Cells(intI, 3) = Hatvany(Cells(intI, 1))
   Cells(intI, 4) = Cells(intI, 2) * Cells(intI, 3)
Next intI
```

Az adatbeolvasás, valamint f(x), g(x) és f(x) * g(x) értékének kiszámítása összevonható, ahogy az alábbi programrészletet mutatja:

```
Do While Not EOF(1)
   intI = intI + 1
   intVeg = intI
   Input #1, intBe
   Cells(intI, 1) = intBe
   Cells(intI, 2) = Exponencialis(intBe)
   Cells(intI, 3) = Hatvany(intBe)
   Cells(intI, 4) = Cells(intI, 2) * Cells(intI, 3)
Loop
```

Ebben az esetben a beolvasáskor nincs szükség az intVeg változó alkalmazására, de az adatok kiírásakor igen. Az adatok kiírásakor For - Next ciklus is alkalmazható, mert ismert a kiírandó adatok száma:

```
Open "4-ki.txt" For Output As #2

For intI = 1 To intVeg
   For intJ = 1 To 4
      If intJ < 4 Then
         Write #2, Cells(intI, intJ),
         Else
         Write #2, Cells(intI, intJ)
      End If
   Next intJ
Next intI

Close #2

End Sub
```

Bár a „Write #2, Cells(intI, intJ),” és „Write #2, Cells(intI, intJ)” parancsok között csak egy vessző az eltérés, ami azt okozza, hogy előbbi esetben a következő érték kiírása is ugyanabba a sorba történik, míg utóbbi esetben az érték kiírása után a program sortörést hajt végre. Végezetül a főprogram által használt függvényeket kell megírni.

```
Function Exponencialis(intBef!) As Single
Exponencialis = intBef * Exp(-2 * intBef)
End Function

Function Hatvany(intBef%) As Long
Hatvany = intBef ^ intBef
End Function
```

A hatványra emelés miatt a Hatvany függvény értéktartományának jóval nagyobbnak kell lennie, mint az intBef változó integer értéktartománya. Ennek a követelménynek megfelel a long adattípus.

6.2.6 Gyakorlatok tömbök alkalmazására

6.2.6.1 Prímszámok feladat

Írjon VBA programot, amelyben InputBox utasítással beolvas egy integer típusú számot (intN), majd a program keresse meg az összes 2 és intN közé eső prímszámot. A talált prímszámokat a Prim() tömb tárolja, majd a tömb tartalmát mentse ki a Primek-intN-ki.txt (pl: Primek-100-ki.txt) file-ba.

Megoldás:

A prímek meghatározása az ún. „eratosthenesi szita” segítségével történhet. Ennek lényege, hogy egy tömbbe beleírják a számokat intN-ig, majd kiveszik az összes számot, ami kettővel osztható. Ezután a tömbben maradt legkisebb szám a 3, ezért a tömbből kiveszik az az összes, hárommal osztható számot és így kell a továbbiakban is folytatni. A legnagyobb lehetséges prímszám intN négyzetgyöke lehet (pl: 529 esetében 23), így a szitálást intN négyzetgyökének egész részének elérésekor kell befejezni.
A változók deklarálása után be kell kérni intN értékét:

```
Sub Primek2()

Dim Prim() As Integer, Munka%()
Dim intIndex As Integer, intJ%, intK%, intN%, intVeg%

intN = InputBox("Kerem, adja meg intN erteket! (intN <= 32767)", "Adatbekeres", 400)
```

Azután meg kell határozni azt a számot, ahol a szitálást be kell fejezni (Sqr függvény négyzetgyököt von, Int függvény a szám egész részét veszi):

```
intVeg = Int(Sqr(intN))
```

Következő lépés a számokat tartalmazó Munka tömb feltöltése:

```
ReDim Munka(intN - 1)

For intJ = 1 To intN - 1
Munka(intJ) = intJ + 1
Next intJ
```

Mivel a tömb első elemének értéke 2, ezért Munka(intN-1) = intN. A lehetséges prímszámokat a tömb a Munka(intVeg-1) = intVeg értékig tartalmazza, ezekkel a számokkal kell a tömbben lévő számokat megvizsgálni, melyre For - Next ciklus is használható (intK ciklusváltozó):

```
For intK = 1 To intVeg - 1
   If Munka(intK) > 0 Then
      For intJ = intK + 1 To intN - 1
         If Munka(intJ) > 0 Then
            If Munka(intJ) Mod Munka(intK) = 0 Then
            Munka(intJ) = 0
            End If
         End If
      Next intJ
   End If
Next intK
```

A Munka(intK) > 0 kitétel azért szükséges, nehogy nullával osszunk, mert ha eredetileg Munka(intK) nem primszam volt, akkor egy korábbi vizsgálat már lenullázta. A prímszámvizsgálat nyilvánvalóan a tömb következő (intK+1) elemétől kezdődhet, erre szintén egy For - Next ciklus használható (intJ ciklusváltozó). Az Munka(intJ) > 0 kitetel azert kell, hogy a kinullázott számokat már ne kelljen vizsgálni. Amennyiben a maradékosztás (Mod parancs) eredménye nulla, Munka(intJ) nem prímszám, értékét le kell nullázni. A következő lépésben a Munka tömb nullától eltérő értékeit kell átmásolni a Prim tömbbe:

```
intIndex = 0

For intJ = 1 To intN - 1
If Munka(intJ) > 0 Then
   intIndex = intIndex + 1
   ReDim Preserve Prim(intIndex)
   Prim(intIndex) = Munka(intJ)
End If
Next intJ
```

Az átmásolás után a Prim tömb elemeinek számát intIndex tartalmazza. A fájlba kiírás történhet For - Next vagy Do - Loop ciklussal is. Do - Loop ciklus alkalmazása esetén a ciklusváltozó értékét minden ciklusban meg kell növelni.

```
Open "Primek2-" & intN & "-ki.txt" For Output As #1
intJ = 0

Do
intJ = intJ + 1
Write #1, Prim(intJ)
Loop While intJ < intIndex

Close #1

End Sub
```

6.2.6.2 Leg nagyobb közös osztó és legkisebb többszörös feladat

Írjon VBA programot, amely először az intPrim tömbbe egymást követő prímszámokat olvas be (első eleme 2), majd kiszámítja az Inputbox paranccsal beadott intSzam1 és intSzam2 (integer típusú) számok legkisebb közös többszörösét (lngLKT, long) és legkisebb közös osztóját (intLKO, integer). Az intTomb (integer) tömb egyes oszlopaiban tárolja intSzam1, intSzam2, lngLKT és intLKO törzstényezős alakjukban szereplő prímszámok hatványkitevőit. A MsgBox utasítással írassa ki intSzam1, intSzam2, lngLKT és intLKO értékét és mentse le a LKT_LKO_intSzam1_intSzam2-ki.txt (pl: LKT_LKO_128_162-ki.txt) fájlba intSzam1, intSzam2, lngLKT és intLKO értékét, valamint intPrim és intTomb tömböket.

Megoldás:

A legnagyobb közös osztó és legkisebb közös többszörös meghatározása során először a felhasználó által megadott számok törzstényezős alakját kell meghatározni (például $24 = 2^3 \times 3$). Ez úgy történhet, hogy a számokat prímszámokkal kell osztani és az így kapott értéket az intMaradek1 és intMaradek2 változókban kell tárolni és a prímszámokkal való osztást addig kell végezni, míg intMaradek1 > 1 és intMaradek2 > 1.

A változók deklarálásakor figyelni kell arra, hogy a legkisebb közös többszörös értéke nagyobb, mint a felhasználó által megadott számok értéke (amennyiben a mindkét szám prímszám, akkor a legkisebb közös többszörös értéke a két szám szorzata), ezért a legkisebb közös többszörös értékét tartalmazó változó gyakran nem lehet integer típusú.

```
Sub LKT_LKO()

Dim intPrim() As Integer, intTomb%()
Dim intI As Integer, intJ%, intLKO%, intMaradek1%, intMaradek2%
Dim intPrimMax%, intPrimVeg%, intSzam1%, intSzam2%
Dim lngLKT As Long
Dim strFnev As String
```

Ezután meg kell kérni a felhasználót, hogy nyissa mega prímszámokat tartalmazó fájlt:

```
MsgBox "Kerem, nyissa meg a primszamokat tartalmazo file-t."

strFnev = Application.GetOpenFilename
Open strFnev For Input As #1
```

Nem ismert, hogy a fájl mennyi prímszámot tartalmaz, ezért az adatok beolvasását a fájl végéig és Do - Loop ciklus alkalmazásával kell végezni. Az intPrim dinamikus tömb méretét minden beolvasáskor meg kell növelni a Redim Preserve paranccsal (A Redim önmagában nem elég, mert akkor a tömb korábbi elemeinek értékét lanulláza a program):

```
intI = 0

Do While Not EOF(1)
   intI = intI + 1
   ReDim Preserve intPrim(intI)
   Input #1, intPrim(intI)
Loop

Close #1
```

Az intI változó megadja, hogy hány prímszámot olvasott be a program, értékét célszerű átadni az intPrimVeg változónak, mert intI ciklusváltozó, így értéke később átírásra kerülhet.

```
intPrimVeg = intI
```

Ezt követően definiálni kell intTomb méretét és a tömböt fel kell tölteni nulla értékekkel. A tömb egyik dimenziója a beolvasott prímszámok darabszáma, míg a másik dimenzió 4 (intSzam1, intSzam2, lngLKT és intLKO törzstényezős alakjainak hatványkitevőinek tárolására)

```
ReDim intTomb(intPrimVeg, 4)

For intI = 1 To intPrimVeg
   For intJ = 1 To 4
   intTomb(intI, intJ) = 0
   Next intJ
Next intI
```

Ezután a felhasználótól bekérhető a két szám, de arra ügyelni kell, hogy a számok nem lehetnek nagyobbak a legnagyobb beolvasott prímszám négyzeténél (lásd Prímszámok feladat):

```
Do
   intSzam1 = InputBox("Kerem, adja meg Szam1 erteket (< 32767)!", "Adatbekeres")
   intSzam2 = InputBox("Kerem, adja meg Szam2 erteket (< 32767)!", "Adatbekeres")
   If intSzam1 > intPrim(intPrimVeg) ^ 2 Or intSzam2 > intPrim(intPrimVeg) ^ 2 Then
      MsgBox "Szam1 es/vagy Szam2 nagyobb, mint a beolvasott legnagyobb primszam négyzete"
   End If
Loop While intSzam1 > intPrim(intPrimVeg) ^ 2 Or intSzam2 > intPrim(intPrimVeg) ^ 2
```

A beolvasott számok értékét át kell adni intMaradek1 és intMaradek2 változóknak:

```
intMaradek1 = intSzam1
intMaradek2 = intSzam2
```

Utána kezdődhet intMaradek1 és intMaradek2 változók prímszámokkal való osztása. Nem lehet előre tudni, hogy a beolvasott prímszámok közül mennyit kell felhasználni az osztások során, ezért Do - Loop ciklust kell alkalmazni:

```
intI = 1
Do While intMaradek1 > 1 Or intMaradek2 > 1
   If intMaradek1 Mod intPrim(intI) = 0 Or intMaradek2 Mod intPrim(intI) = 0 Then
      If intMaradek1 Mod intPrim(intI) = 0 Then
         intMaradek1 = intMaradek1 / intPrim(intI)
         intTomb(intI, 1) = intTomb(intI, 1) + 1
         Else
         intMaradek2 = intMaradek2 / intPrim(intI)
         intTomb(intI, 2) = intTomb(intI, 2) + 1
      End If
      Else
      intI = intI + 1
   End If
Loop
```

A külső If … then - End If szerkezet megvizsgálja, hogy intMaradek1 vagy intMaradek2 osztható-e az adott prímszámmal. Amennyiben igen, akkor a belső if … then - End If szerkezetével megnöveli eggyel intTomb megfelelő elemének az értékét, míg ellentétes esetben a ciklusváltozó értékét növeli meg. Az egymásba ágyazott If … then - End If szerkezetek lehetővé teszik, hogy ugyanazzal a prímszámmal több-ször is osztani lehessen. Természetesen az alábbi szerkezet is alkalmazható:

```
intI = 1
Do While intMaradek1 > 1 Or intMaradek2 > 1
   If intMaradek1 Mod intPrim(intI) = 0 Or intMaradek2 Mod intPrim(intI) = 0 Then
      If intMaradek1 Mod intPrim(intI) = 0 Then
         intMaradek1 = intMaradek1 / intPrim(intI)
         intTomb(intI, 1) = intTomb(intI, 1) + 1
      End If
      If intMaradek2 Mod intPrim(intI) = 0 Then
         intMaradek2 = intMaradek2 / intPrim(intI)
         intTomb(intI, 2) = intTomb(intI, 2) + 1
      End If
      Else
      intI = intI + 1
   End If
Loop
```

Az intI változó most azt adja meg, hogy az intPrim tömbbe beolvasott prímszámok közül mennyit hasz-nált fel a program addíg, amíg intMaradek1 = 1 és intMaradek2 = 1 lett, értékét célszerű átadni az intPrimMax változónak.

```
intPrimMax = intI
```

A törzsténye zős alakok hatványkitevőinek meghatározása után kiindulási értéket kell adni intLKO és lngLKT változóknak, valamint intI ciklusváltozónak. Mivel intLKO és lngLKT értékét szorzással lehet megkapni a kiindulási érték csak 1 lehet.

```
intLKO = 1
lngLKT = 1
intI = 1
```

Az intTomb változóban lévő értékek segítségével megadható intLKO és lngLKT hatványkitevői (a kisebb érték intLKO, a nagyobb lngLKT értékének meghatározásához szükséges). Mivel intPrimMax értéke ismert, ezért mind Do - Loop ciklus, mind For - Next ciklus alkalmazható:

```
Do
   If intTomb(intI, 1) > intTomb(intI, 2) Then
      intTomb(intI, 3) = intTomb(intI, 1)
      intTomb(intI, 4) = intTomb(intI, 2)
      Else
      intTomb(intI, 3) = intTomb(intI, 2)
      intTomb(intI, 4) = intTomb(intI, 1)
   End If
   lngLKT = lngLKT * intPrim(intI) ^ intTomb(intI, 3)
   intLKO = intLKO * intPrim(intI) ^ intTomb(intI, 4)
   intI = intI + 1
Loop While intI <= intPrimMax
```

Ezt követi intSzam1, intSzam2, lngLKT és intLKO értékének képernyőre és fájlba írása:

```
MsgBox "Szam1: " & intSzam1 & ", " & "Szam2: " & intSzam2 & ", " & "LKT: " & lngLKT & ", " & "LKO: " &
intLKO

Open "LKT_LKO_" & intSzam1 & "_" & intSzam2 & "-ki.txt" For Output As #2

Write #2, "Primek", "Szam1", "Szam2", "LKT", "LKO"
Write #2, "---", intSzam1, intSzam2, lngLKT, intLKO
```

A program a fájlba azt is beleírja, hogy a fenti négy változó, mely prímszámok (intPrim(intI)) hanyadik hatványát (intTomb(intI,1-től 4-ig)) tartalmazzák törzstényezős szorzatukban:

```
intI = 1

Do While intI <= intPrimMax
Write #2, intPrim(intI),
Write #2, intTomb(intI, 1), intTomb(intI, 2), intTomb(intI, 3), intTomb(intI, 4)
intI = intI + 1
Loop

Close #2

End Sub
```

6.2.6.3 Térfogatszámolás feladat

A korábban megírt programot úgy kell módosítani, hogy ciklusokat, tömböket, függvényeket és a szubrutint is tartalmazzon, ezért az alábbiakat kell figyelembe venni:

- R, V, A és f értékeket lehet tömbökben tárolni, míg strDontes és sngPi változókat nem célszerű (lehet 1 elemes tömböket deklarálni, de nincs értelme),
- az if then - goto ujra szerkezetek helyett ciklusokat kell alkalmazni. Mivel nem lehet tudni, hogy a ciklusnak hányszor kell lefutnia, ezért Do - Loop ciklust kell használni. A blokkdiagramon látható, hogy a feltételek (R_1>R_2, új számolás) vizsgálata a ciklus végén történik, ezért hátul tesztelő ciklusra van szükség,
- R, V, A és f értékeket tartalmazó tömbök elemszáma ismert (elemszám: 2, 3, 3, 3), így a tömbökkel végzett műveleteknél For - Next ciklust is lehet használni, melyhez viszont az „i” ciklusváltozó deklarálása szükséges.

A fentieket figyelembe véve a változók deklarálása és π értékének kiszámolása a következőképpen történhet:

```
Sub Terfogatszamolas2()

Dim sngR!(2), sngV!(3), sngA!(3), sngf!(3)
Dim i%
Dim strDontes As String
Dim sngPi As Single

sngPi = 4 * Atn(1)
```

Ezt követi a két Do - Loop ciklus megnyitása és a két sugárérték beolvasása:

```
Do
Do
sngR(1) = InputBox("R1 = ?", "Adatbekeres", 2)
sngR(2) = InputBox("R2 = ? (kisebb legyen, mint R1)", "Adatbekeres", 1)
```

Bár For - Next ciklussal is megoldható az adatok beolvasása, de nem célszerű, mert a felhasználónak kiírandó szöveg jelentősen eltér, ami miatt if - else - End If szerkezetet kellene használni, de ez bonyolultabbá tenné a programot. A következő lépés a R_1>R_2 feltétel vizsgálata, az első ciklus bezárása:

```
Loop While sngR(2) >= sngR(1)
```

Ezután a térfogat- és felületértékek számítása következik:

```
For i = 1 To 2
   sngV(i) = Terfogat(sngR(i), sngPi)
   sngA(i) = Felulet(sngR(i), sngPi)
Next i

sngV(3) = sngV(1) - sngV(2)
sngA(3) = sngA(1) + sngA(2)
```

A térfogat, illetve a felület számolásához a „Terfogat”, illetve „Felulet” függvények használhatóak, melyeknek a program átadja a sugár és π értékét. Ez a programrész azonban eltér a blokkdiagramon láthatótól. A blokkdiagram szerint a számolásnak a V_1, V_2, V_3, A_1, A_2, A_3 sorrendben kellene történnie, míg a fenti programrészlet V_1, A_1, V_2, A_2, V_3, A_3 sorrendet alkalmaz. A blokkdiagramnak az alábbi programrészlet felelne meg:

```
For i = 1 To 2
   sngV(i) = Terfogat(sngR(i), sngPi)
Next i

sngV(3) = sngV(1) - sngV(2)

For i = 1 To 2
   sngA(i) = Felulet(sngR(i), sngPi)
Next i

sngA(3) = sngA(1) + sngA(2)
```

Ebben az esetben viszont két For - Next ciklust kell alkalmazni, de az első megoldással az egyik For - Next ciklus elhagyható. Most következik a fajlagos felület értékek kiszámolása és az eredmények kiírása:

```
For i = 1 To 3
   sngf(i) = sngA(i) / sngV(i)
   MsgBox "V" & i & " = " & sngV(i) & ", A" & i & " = " & sngA(i) & ", f" & i & "= " & sngf(i)
Next i
```

A blokkdiagram f_1, f_2, f_3, Kiír1, Kiír2, Kiír3 sorrendet, míg az előző programrészlet f_1, Kiír1, f_2, Kiír2, f_3, Kiír3 sorrendet alkalmaz. Végül az „Uj szamolas” feltétel vizsgálata, a második ciklus bezárása és a program vége következik:

```
strDontes = InputBox("Szeretne uj szamolast vegezni? (i = igen, bármi más = nem)", "Adatbekeres", "i")
Loop While strDontes = "i"

MsgBox "Program vege"

End Sub
```

A „Terfogat” és „Felulet” függvények sngRbe és sngPibe változóiba adja át a főprogram sngR(1), sngR(2) és sngPi értékét. Nem okoz gondot, hogy mindkét függvény egy-egy sngRbe és sngPibe változót használ, mert az azonos nevű változók alkalmazási területe nem fed át.

```
Function Terfogat(sngRbe!, sngPibe!) As Single
   Terfogat = 4 * sngRbe ^ 3 * sngPibe / 3
End Function

Function Felulet(sngRbe!, sngPibe!) As Single
   Felulet = 4 * sngRbe ^ 2 * sngPibe
End Function
```

Ha a két függvény helyett egy szubrutint használ a program, akkor a főprogramban a

```
For i = 1 To 2
   sngV(i) = Terfogat(sngR(i), sngPi)
   sngA(i) = Felulet(sngR(i), sngPi)
Next i
```

programrészlet helyett a

```
For i = 1 To 2
   Call Terfogat_Felulet(sngR(i), sngPi, sngV(i), sngA(i))
Next i
```

programrészletet kell alkalmazni, míg maga a „Terfogat_Felulet” szubrutin a következőképpen néz ki:

```
Sub Terfogat_Felulet(sngRbe!, sngPibe!, sngVki!, sngAki!)
   sngVki = 4 * sngRbe ^ 3 * sngPibe / 3
   sngAki = 4 * sngRbe ^ 2 * sngPibe
End Sub
```

A szubrutin az sngVki és sngAki változók értékét adja át a főprogram sngV(i) és sngA(i) változóinak.

6.2.6.4 Öröknaptár feladat

A korábban megírt programot úgy kell változtani, hogy a Select Case - End Select utasítást kell kiváltani egy tömbbel, mely a hét napjait tartalmazza.

```
Sub Oroknaptar_tomb()

Dim datReferencia As Date, datKerdesesNap As Date
Dim intMaradek As Integer
Dim sngKulonbseg As Single
Dim Napok(6) As String
```

A Napok tömb elemeit fel kell tölteni a hét napjaival. Mivel a maradékosztás eredménye nulla is lehet, ezért a napokat célszerű a tömb nulladik elemétől a hatodik elemig beadni.

```
Napok(0) = "hetfo"
Napok(1) = "kedd"
Napok(2) = "szerda"
Napok(3) = "csutortok"
Napok(4) = "pentek"
Napok(5) = "szombat"
Napok(6) = "vasarnap"

datReferencia = #8/31/2009#
datKerdesesNap = InputBox("Kerem, adja meg a kerdeses nap datumat!", "Adatbekeres", #1/1/2009#)
sngKulonbseg = datKerdesesNap - datReferencia
intMaradek = sngKulonbseg Mod 7
```

Amennyiben intMaradek értéke negatív szám, akkor a változó értékét héttel meg kell növelni. Ezután következhet a kérdéses nap kiírása.

```
If intMaradek < 0 Then intMaradek = intMaradek + 7

MsgBox "A kerdeses nap " & Napok(intMaradek) & " volt/lesz."

End Sub
```

Másik lehetséges megoldás, amikor a variant típusú Napok tömb elemeit felsorolásszerűen adják meg:

```
Dim Napok As Variant

Napok = Array("hetfo", "kedd", "szerda", "csutortok", "pentek", "szombat", "vasarnap")
```

A program többi része megegyezik a fentebb leírtakkal.

6.2.6.5 Egyenletmegoldás feladat

Írja meg az „Egyenletmegoldas” nevű VBA programot, amely először beolvassa az alábbi két egyenletrendszer közül az egyik együtthatóit az „Egyenletek.txt” vagy az „Egyenletek2.txt” file-ból egy 3×4 nagyságú tömbbe, majd az egyenletrendszert megoldja és a tömb tartalmát egyrészt a munkalap A1:D3 celláiba, másrészt az „Egyenletek-megoldas.txt” file-ba írja ki. A program írásakor az sngHanyados, sngTomb(3,4) (mindkettő single), az intI, intJ, intK (mind integer) változókat alkalmazza.

$$\begin{aligned} 6x+2y-3z&=-4\\ -x+3y+4z&=-2\\ 2x-3y-3z&=9 \end{aligned}$$

$$\begin{aligned} 2x+-6y-8z&=4\\ 6x+2y-3z&=-4\\ 2x-3y-3z&=9 \end{aligned}$$

egyenletrendszerek

```
6,2,-3,-4
-1,3,4,-2
2,-3,-3,9
```

Egyenletek.txt

	A	B	C	D
1	1	0	0	3
2	0	1	0	-5
3	0	0	1	4

```
1,0,0,3
0,1,0,-5
0,0,1,4
```

megoldás Excel munkalapon és fájlban

Megoldás:

A változók deklarálása után meg kell nyitni az Egyenletek.txt fájlt, majd a fájlban található értékekkel fel kell tölteni az sngTomb tömböt, melyre két egymásba ágyazott For - Next ciklust kell használni.

```
Sub Egyenletmegoldas()

Dim intI As Integer, intJ%, intK%
Dim sngHanyados As Single
Dim sngTomb(3, 4) As Single

Open "Egyenletek.txt" For Input As #1

For intI = 1 To 3
   For intJ = 1 To 4
   Input #1, sngTomb(intI, intJ)
   Cells(intI, intJ) = sngTomb(intI, intJ)
   Next intJ
Next intI
Close #1
```

Ezután az egyenlet együtthatóit tartalmazó tömbrész (3 × 3) diagonálisa alatti számokat kell kinullázni. Először az sngHanyados változó segítségével meg kell határozni, hogy a tömb egyes sorainak elemeit mekkora értékkel kell megszorozni a sorok egymásból való kivonása előtt. Az intI a kivonandó sor indexe, az intJ ahhoz a sorhoz tartozik, melyből a kivonás történik, míg intK az egyes sorok elemeihez tartozik (a jobboldalon lévő ábra a program működését mutatja, de ezt a program a működése során nem írja ki).

	A	B	C	D
1	6	2	-3	-4
2	-1	3	4	-2
3	2	-3	-3	9
4				
5	1. lépés			
6	6	2	-3	-4
7	6	-18	-24	12
8	6	-9	-9	27
9				
10	2.lépés			
11	6	2	-3	-4
12	0	-20	-21	16
13	0	-11	-6	31
14				
15	3. lépés			
16	6	2	-3	-4
17	0	-20	-21	16
18	0	-20	-10,909	56,3636
19				
20	4. lépés			
21	6	2	-3	-4
22	0	-20	-21	16
23	0	0	10,0909	40,3636

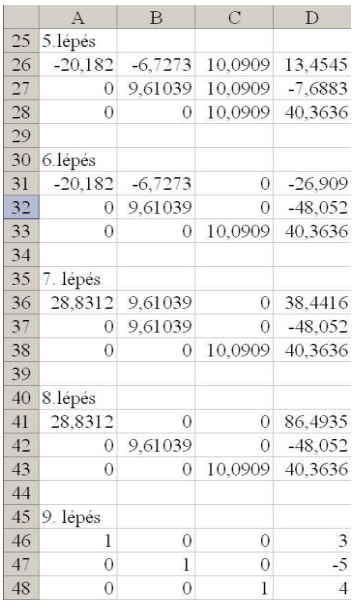
```
For intI = 1 To 2
   For intJ = intI + 1 To 3
      sngHanyados = sngTomb(intI, intI) / sngTomb(intJ, intI)
      For intK = 1 To 4
            sngTomb(intJ, intK) = sngHanyados * sngTomb(intJ, intK)
            sngTomb(intJ, intK) = sngTomb(intJ, intK) - sngTomb(intI, intK)
      Next intK
   Next intJ
Next intI
```

Ezt követi a diagonális feletti számok kinullázása. A számolás során mindenképp a lépésköz értéke -1 (Step -1), mert a magasabb számú soroktól kell visszafelé haladni az első sor felé.

```
For intI = 3 To 2 Step -1
   For intJ = intI - 1 To 1 Step -1
      sngHanyados = sngTomb(intI, intI) / sngTomb(intJ, intI)
      For intK = 1 To 4
            sngTomb(intJ, intK) = sngHanyados * sngTomb(intJ, intK)
            sngTomb(intJ, intK) = sngTomb(intJ, intK) - sngTomb(intI, intK)
      Next intK
   Next intJ
Next intI
```

Ezután a diagonális értékeit kell 1-re normálni.

```
For intI = 1 To 3
   sngHanyados = sngTomb(intI, intI)
   sngTomb(intI, intI) = sngTomb(intI, intI) / sngHanyados
   sngTomb(intI, 4) = sngTomb(intI, 4) / sngHanyados
Next intI
```



	A	B	C	D
25	5.lépés			
26	-20,182	-6,7273	10,0909	13,4545
27	0	9,61039	10,0909	-7,6883
28	0	0	10,0909	40,3636
29				
30	6.lépés			
31	-20,182	-6,7273	0	-26,909
32	0	9,61039	0	-48,052
33	0	0	10,0909	40,3636
34				
35	7. lépés			
36	28,8312	9,61039	0	38,4416
37	0	9,61039	0	-48,052
38	0	0	10,0909	40,3636
39				
40	8.lépés			
41	28,8312	0	0	86,4935
42	0	9,61039	0	-48,052
43	0	0	10,0909	40,3636
44				
45	9. lépés			
46	1	0	0	3
47	0	1	0	-5
48	0	0	1	4

Végezetül a sngTomb tömb értékeit kell fájlba írni. Az If - End If szerkezet teszi lehetővé, hogy a következő kiírandó értéket ugyanabba vagy egy új sorba kell kiírni (van, illetve nincs vessző/pontosvessző a write parancs sorának végén).

```
Open "Egyenletek-megoldas.txt" For Output As #2
For intI = 1 To 3
   For intK = 1 To 4
   Cells(intI, intK) = sngTomb(intI, intK)
   If intK < 4 Then
      Write #2, sngTomb(intI, intK);
      Else
      Write #2, sngTomb(intI, intK)
   End If
   Next intK
Next intI
Close #2

End Sub
```

6.3 Önállóan megoldandó gyakorló feladatok

6.3.1 Feladatok

6.3.1.1 For-Next ciklus

6.3.1.1.1 Szorzótábla feladat

Írassa ki a 12×12 szorzótáblát egy üres munkalapra.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1														
2			1	2	3	4	5	6	7	8	9	10	11	12
3			2	4	6	8	10	12	14	16	18	20	22	24
4			3	6	9	12	15	18	21	24	27	30	33	36
5			4	8	12	16	20	24	28	32	36	40	44	48
6			5	10	15	20	25	30	35	40	45	50	55	60
7			6	12	18	24	30	36	42	48	54	60	66	72
8			7	14	21	28	35	42	49	56	63	70	77	84
9			8	16	24	32	40	48	56	64	72	80	88	96
10			9	18	27	36	45	54	63	72	81	90	99	108
11			10	20	30	40	50	60	70	80	90	100	110	120
12			11	22	33	44	55	66	77	88	99	110	121	132
13			12	24	36	48	60	72	84	96	108	120	132	144

Írjon programot, ami a szorzótábla alsó háromszög részletét jeleníti meg! A sorok számát Inputbox utasítással kérdezze meg.

	1	2	3	4	5	6	7	8	9	10	11
1											
2											
3		1									
4		2	4								
5		3	6	9							
6		4	8	12	16						
7		5	10	15	20	25					
8		6	12	18	24	30	36				
9		7	14	21	28	35	42	49			
10		8	16	24	32	40	48	56	64		
11		9	18	27	36	45	54	63	72	81	
12		10	20	30	40	50	60	70	80	90	100

6.3.1.1.2 Pascal háromszög feladat

Írassa ki a Pascal háromszög egy részletét a munkalapra! Inputbox utasítással kérdezzen rá, mennyi sort írjon ki belőle a program! (http://hu.wikipedia.org/wiki/Pascal-h%C3%A1romsz%C3%B6g)

	1	2	3	4	5	6
1		1				
2		1	1			
3		1	2	1		
4		1	3	3	1	
5		1	4	6	4	1

Írassa ki a Pascal háromszög egy részletét a munkalapra! Inputbox utasítással kérdezzen rá, mennyi sort írjon ki belőle a program, de ezúttal az alábbi ábrán láthatóan, szépen elrendezve jelenjen meg a háromszög!

	1	2	3	4	5	6	7	8	9	10	11	12
1							1					
2						1		1				
3					1		2		1			
4				1		3		3		1		
5			1		4		6		4		1	
6		1		5		10		10		5		1

6.3.1.2 Do-Loop ciklus

6.3.1.2.1 Véletlen sorsolás

A program véletlen számokat sorsoljon ki és ezeket írja egymás alatti cellákba, amíg egy megadott határértéknél nagyobbat nem sorsol ki, ekkor hagyja abba a sorsolást és írja ki a kisorsolt számok összegét és átlagukat. A határértéket a sorsolás megkezdése előtt egy Inputbox utasítással kérdezze meg (az alapérték legyen 0.95)

Véletlen számot az Rnd utasítással tud generálni (pl: X = rnd), mely utasítás [0,1] intervallumba eső véletlenszerű érteket add.

	A	B	C	D	E	F
1	n	x		N	összeg	átlag
2	1	0,910964		4	2,812949	0,703237
3	2	0,226866				
4	3	0,695116				
5	4	0,980003				
6	5	0,524868				
7	6	0,767112				
8	7	0,053505				
9	8	0,592458				
10	9	0,4687				
11	10	0,298165				
12	11	0,622697				
13	12	0,647821				
14	13	0,263793				
15	14	0,279342				
16	15	0,829802				
17	16	0,824602				
18	17	0,589163				
19	18	0,986093				

6.3.1.2.2 Faktoriális számítása

Írjon programot, ami egy szám faktoriálisát számolja ki! A számot Inputbox utasítással kérdezze meg a felhasználótól! A számolás után az eredményt a Msgbox utasítással írja ki a program. Végül a program kérdezze meg szeretnénk-e új számolást, ha a válasz „i”, akkor kezdje az egészet elölről.

6.3.1.3 Szubrutin alkalmazása

6.3.1.3.1 Oszthatóság vizsgálata

Töltsön fel egy 10×10 mezőt véletlen számokkal, melyek értéke 1-100 közé esik (Int(Rnd * 100 + 1). Majd a hárommal osztható számokat tartalmazó cellákat színezze pirosra, az öttel oszthatókat kékre, a héttel oszthatókat zöldre. A színezéshez használjon szubrutint, amelynek a paraméterei: egy string (a szín megadása), két integer (a színezendő cella koordinátái).

Egy cella színét a következő utasítással változtathatja meg:

Cells(y, x).Interior.Color = RGB(0, 200, 0) zöld

Cells(y, x).Interior.Color = RGB(200, 0, 0) piros

Cells(y, x).Interior.Color = RGB(0, 0, 200) kék

	A	B	C	D	E	F	G	H	I	J
1	79	6	52	76	51	33	98	81	68	91
2	88	42	13	66	80	70	41	2	17	17
3	51	41	11	28	65	85	50	19	90	38
4	33	78	22	45	24	88	62	38	69	87
5	59	94	52	34	87	26	26	13	35	1
6	75	85	28	71	41	82	75	44	8	42
7	34	72	32	80	16	60	96	25	94	13
8	99	64	60	91	58	25	87	8	44	76
9	25	38	40	53	28	59	21	8	90	11
10	66	91	24	95	85	45	50	77	84	35

6.3.1.4 Összetett feladatok

6.3.1.4.1 Kockadobás feladat

Dobjon a számítógép két hatoldalú kockával és a dobások eredményét két, egymás melletti oszlopba írja ki. Ezt addig ismételje, míg ötször nem dob azonosat a két kockával. Ezután a dobások számát írja ki egy külön ablakba (Msgbox utasítás)

A dobásokat külön függvénnyel (function) számolja ki. Véletlen egész számokat az Int((Rnd × 6) + 1) utasítással tud generálni.

A kettős dobásokat tartalmazó cellákban a szöveg színét a jobb láthatóság érdekében a következő utasítás megfelelő használatával pirosra változtathatja: Cells(x, y).Font.Color = RGB(255, 0, 0)

	A	B
1	6	6
2	4	4
3	4	1
4	2	2
5	1	1
6	3	4
7	4	5
8	1	3
9	1	2
10	4	4
11	4	2
12	6	5
13	4	3
14	5	1
15	1	2
16	5	6
17	3	5
18	5	1
19	5	3
20	2	4
21	6	6

	A	B	C	D	E
1	5		legnagyobb dobás=		15
2	4				
3	4				
4	2				
5	2				
6	5				
7	1				
8	5				
9	5				
10	5				
11	1				
12	3				
13	6	5			
14	3				
15	6	6	1		
16	6	3			
17	4				
18	5				
19	1				
20	4				
21	3				
22	2				
23	4				
24	4				
25	2				
26	2				
27	5				
28	5				
29	4				
30	6	6	2		
31	5				
32	6	2			

6.3.1.4.2 Kockadobás újra feladat

Dobjon a program hatoldalú kockával százszor, miközben a dobások értékét egymás alatti cellákba írja ki. Hatos dobás esetén dobjon újra és az értéket a mellette lévő cellába írja be, ismételt hatos dobás esetén az új számot a harmadik oszlopba, így tovább még végül nem hatost dob.

Az egy sorba írt számokat egy dobásnak véve (pl: két hatos és egy kettes dobás értéke: 14), keresse meg a legnagyobb dobást és a dobást tartalmazó sor számát írja ki.

6.3.1.4.3 Táblázat feladat

Egy táblázatba írasson ki véletlen egész számokat, a táblázat sorainak és oszlopainak számát Inputbox utasítással kérdezze meg. A kiírt véletlen számok maximális értéke legyen oszlop szám +1 (például a harmadik oszlopban 1, 2, 3, 4 a lehetséges értékek). Az első sorban a lehetséges maximumértékek szerepeljenek. A véletlen értékek számolásához írjon függvényt, amely bemenő értéke a lehetséges maximum.

A páros számokat színezze pirosra! (az Int utasítást használhatja)

	A	B	C	D	E	F	G	H	I	J
1	max=	3	4	5	6	7	8	9	10	11
2	1	3	3	3	3	4	1	2	5	5
3	1	1	3	3	3	3	1	7	5	4
4	2	3	4	2	1	1	3	5	5	10
5	1	2	4	3	3	6	2	4	3	9
6	2	1	1	2	2	4	1	2	4	4
7	1	3	3	1	4	6	5	7	1	8
8	1	2	4	2	4	5	6	9	6	3
9	2	2	2	5	1	6	5	5	2	1
10	1	2	3	1	6	1	7	1	10	1
11	1	1	3	2	4	2	1	7	1	4

6.3.2 Megoldások

6.3.2.1 For-Next ciklus

6.3.2.1.1 Szorzótábla feladat

Teljes szorzótábla kiíratása:

```
Sub szorzotábla()
Dim i As Integer
Dim k As Integer

For i = 1 To 12
   For k = 1 To 12
     Cells(k + 1, i + 2) = i * k
   Next k
Next i

End Sub
```

Alsó háromszög kiíratása:

```
Sub haromszog()

Dim i As Integer
Dim k As Integer
Dim m As Integer

m = InputBox("Mennyi?", "Kérdés", 5)

For i = 1 To m
   For k = 1 To i
      Cells(i + 2, k + 1) = i * k
   Next k
Next i

End Sub
```

6.3.2.1.2 Pascal háromszög feladat

Pascal háromszög kiíratása:

```
Sub Pascal1()

Dim i%, k%, n%

n = InputBox("Mennyi sort?", "kérdés", 4)
Cells(1, 2) = 1

For i = 2 To n + 1
   For k = 2 To i + 1
      Cells(i, k) = Cells(i - 1, k - 1) + Cells(i - 1, k)
   Next k
Next i

End Sub
```

Pascal háromszög szebb elrendezésű kiíratása:

```
Sub Pascal2()

Dim i%, k%, n%, x%

Cells.HorizontalAlignment = xlCenter

n = InputBox("mennyi?", "kérdés", 4)
Cells(1, n + 1) = 1

For i = 2 To n
   For k = 1 To i
      x = k * 2 - i + n
      Cells(i, x) = Cells(i - 1, x - 1) + Cells(i - 1, x + 1)
   Next k
Next i

End Sub
```

6.3.2.2 Do-Loop ciklus

6.3.2.2.1 Véletlen sorsolás

```
Sub Veletlen()

Dim x!, osszeg!, limit!
Dim szamlalo%

Cells(1, 1) = "n"
Cells(1, 2) = "x"
Cells(1, 4) = "N"
Cells(1, 5) = "összeg"
Cells(1, 6) = "átlag"


limit = InputBox("A határ érték (0 - 1)", "Kérdés", "0,95")
szamlalo = 0
osszeg = 0

Do
   x = Rnd
   szamlalo = szamlalo + 1
   osszeg = osszeg + x
   Cells(szamlalo + 1, 1) = szamlalo
   Cells(szamlalo + 1, 2) = x
Loop Until x > limit

Cells(2, 4) = szamlalo
Cells(2, 5) = osszeg
Cells(2, 6) = osszeg / szamlalo

End Sub
```

6.3.2.2.2 Faktoriális számítása

```
Sub Faktoriális()

Dim n%, i%
Dim valasz$
Dim f As Double

Do
   n = InputBox("n=?", "Faktor számolás", 2)
   f = 2
   For i = 2 To n
      f = f * i
   Next i
   MsgBox " " & n & "!=" & f
   valasz = InputBox("Újabb számolás? (i/n)", "Kérdés", "i")
Loop While valasz = "i"

End Sub
```

6.3.2.3 Szubrutin alkalmazása

6.3.2.4 Oszthatóság vizsgálata

```
Sub szinezo()

Dim i%, k%, n%

For i = 1 To 10
   For k = 1 To 10
      n = Int(Rnd * 100 + 1)
      Cells(i, k) = n
      If Int(n / 3) = n / 3 Then szin("k", i, k)
      If Int(n / 5) = n / 5 Then szin("z", i, k)
      If Int(n / 7) = n / 7 Then szin("p", i, k)
   Next k
Next i

End Sub


Sub szin(s As String, y%, x%)

If s = "z" Then Cells(y, x).Interior.Color = RGB(0, 200, 0)
If s = "k" Then Cells(y, x).Interior.Color = RGB(0, 0, 200)
If s = "p" Then Cells(y, x).Interior.Color = RGB(200, 0, 0)

End Sub
```

6.3.2.5 Összetett feladatok

6.3.2.5.1 Kockadobás feladat

```
Sub kockak()
Dim n%, db%

db = 0
n = 0
Do
   n = n + 1
   Cells(n, 1) = dobas
   Cells(n, 2) = dobas
   If Cells(n, 1) = Cells(n, 2) Then
   db = db + 1
      Cells(n, 1).Font.Color = RGB(255, 0, 0)
      Cells(n, 2).Font.Color = RGB(255, 0, 0)
   End If

Loop Until db > 5

MsgBox "A dobások száma: " & n

End Sub


Function dobas() As Integer
   dobas = Int((Rnd * 6) + 1)
End Function
```

6.3.2.5.2 Kockadobás újra feladat

```
Sub kockak()
Dim n%, utolso%, i%, db%, max%, osszeg%

Cells(1, 3) = "legnagyobb dobás="

For i = 1 To 100
   utolso = dobas
   Cells(i, 1) = utolso
   osszeg = utolso
   db = 1
   Do While utolso = 6
      utolso = dobas
      db = db + 1
      Cells(i, db) = utolso
      osszeg = osszeg + utolso
   Loop
   If max < osszeg Then max = osszeg
Next i

Cells(1, 5) = max

End Sub
```

```
Function dobas() As Integer
   dobas = Int((Rnd * 6) + 1)
End Function
```

6.3.2.5.3 Táblázat feladat

```
Sub tabla()
Dim i%, k%
Dim sor%, oszlop%

sor = InputBox("sorok száma?", "Adatbekérés", 5)
oszlop = InputBox("oszlopok száma?", "Adatbekérés", 6)

For k = 1 To oszlop
   Cells(1, k) = k + 1
   For i = 1 To sor
      Cells(i + 1, k) = veletlen(k + 1)
      If Int(Cells(i + 1, k) / 2) = Cells(i + 1, k) / 2 Then
         Cells(i + 1, k).Font.Color = RGB(250, 0, 0)
      End If
   Next i
Next k

Cells(1, 1) = "max="
End Sub


Function veletlen(max As Integer) As Integer
   veletlen = Int((Rnd * max) + 1)
End Function
```

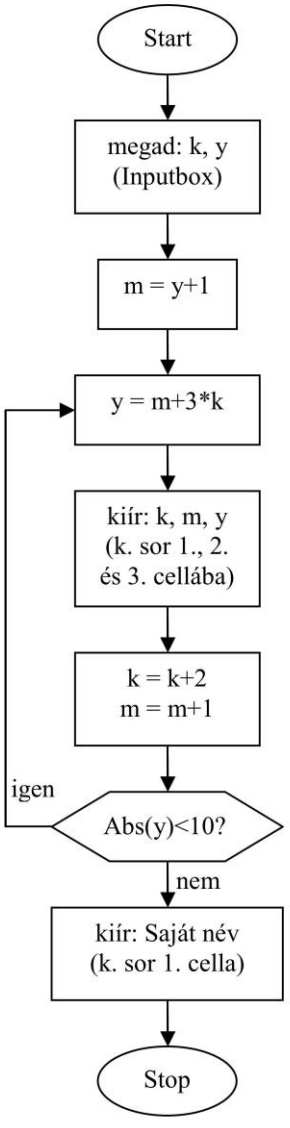
6.4 Korábbi zárthelyi feladatsorok

6.4.1 2009 tavaszi feladatsorok

6.4.1.1 1. feladatsor

1. diger – 2009. tavasz

1. Írjon VBA programot az alábbi blokkdiagram alapján! A „k” változót Integer, „y” és „m” változókat Double típusúnak definiálja! Az InputBox paranccsal történő adatbekérdezés során „k” értéke legyen 1, míg „y” -6.5! Futtassa le a programot! (6. pont)



2. Atomemissziós módszernél az intenzitás (I) és a koncentráció (c) között az $I = a \times c^b$ összefüggés áll fenn, ahol „a” és „b” konstansok. A „c (mg/dm^3)” oszlopban megadott koncentrációk esetén a mért intenzitásokat az „$I_{mért}$” oszlop tartalmazza. Az „a” és „b” konstans értékét tartalmazó M2 és M3 cellák segítségével számítsa ki, hogy a = 0.1 és b = 1 esetben mekkora intenzitást kellett volna mérni („$I_{számolt}$” oszlop). Az „$(I_m-I_{sz})^2$” oszlopban számítsa ki az adott koncentrációhoz tartozó mért és számított intenzitások különbségének a négyzetét, majd a kapott értékek összegét számítsa ki a J8 cellában. Végül a „Solvert”-t használva úgy számítson ki új „a” és „b” értékeket, hogy J8 cella értéke a lehető legkisebb legyen. A megoldás megtalálásakor a Solverrel szúrassa be „A kiszámított értékekkel” az eredményjelentést tartalmazó munkalapot. (4 pont)

	G	H	I	J	K	L	M	N
1	c (mg/dm^3)	$I_{mért}$	$I_{számolt}$	$(I_m-I_{sz})^2$				
2	0,00	0,000				a =	0,100	
3	1,00	0,090				b =	1,000	
4	2,20	0,202						
5	3,10	0,273						
6	5,00	0,440						
7								
8			Összesen =					
9								

6.4.1.2 2. feladatsor

2. diger – 2009. tavasz

1. Adott az $f(k)=3/(k^2+5k+6)$ függvény. Írja meg a „Megoldas1” VBA programot a $\text{Sum} = \sum_{k=1}^{m} f(k) = f(1) + f(2) + ... + f(m)$ összeg kiszámítására. A programban InputBox-szal olvastassa be m és p értékét, mely utóbbi az a legkisebb k érték, amelytől kezdődően a program az összeg (Sum) értékét is kiíratja.

   A programban használjon For-Next ciklusutasítást, m, p és k változókat Integer-nek, Sum-ot Double-nak definiálja. Az f(k) értékét „Function”-nal számolja ki (függvény neve: f, single típusú)! A program kiíratása a „Munka1” munkalapon történjen és az itt megadott képnek (mely m=5 és p=3 esetét mutatja) megfelelő szerkezetű eredményt adja! (6 pont)

	A	B	C
1	k	f(k)	Sum
2	1	0,25	
3	2	0,15	
4	3	0,1	0,5
5	4	0,07143	0,57143
6	5	0,05357	0,625
7			
8	p =	3	
9	m =	5	

2. Hozzon létre új általános modult, majd abban írja át az 1. feladatban létrehozott programot úgy („Megoldas2”), hogy a For-Next ciklus helyett Do While-Loop ciklust, az f „Function” helyett pedig f „Sub”-ot alkalmazzon! Az f „Sub” által kiszámított értéket a „Megoldas2” programban az fki változó (single típusú) tárolja! A programot a „Munka2” munkalapon futtassa. (3 pont)
3. A feladatot a For-Next, vagy a Do-Loop ciklus alkalmazásával oldaná meg akkor, ha a program nem a k = m határig, hanem addig futna, amíg a Sum <= 0.6 feltétel teljesül? Indokolja meg válaszát! (1 pont)

6.4.1.3 3. feladatsor

3. diger – 2009. tavasz

1. Írja meg a „Haromszog” nevű VBA programot, amely kiszámolja a térben elhelyezkedő három pontot (A, B, C) összekötő AB, AC és BC vektorok ismeretében a háromszög egyes oldalainak hosszúságát. A program először For-Next ciklus alkalmazásával olvassa be az „AB.txt” és „AC.txt” file-ból az AB és AC vektorok 3-3 adatát az „intAB” és „intAC” (integer típusú) tömbökbe. A program utána számolja ki a BC vektor („intBC”, integer) paramétereit, amelyek AC és AB vektorok megfelelő paramétereinek a különbségei. Majd a program a „Hossz” function (single típusú, bemenő paraméterek a megfelelő tömbök adatai) alkalmazásával számolja ki az egyes oldalak hosszúságát (sngAB, sngAC és sngBC, mind single típusú) az alábbi összefüggés segítségével:

$$\text{Hosszúság} = \sqrt{(x^2 + y^2 + z^2}\ ,$$

ahol x, y és z az adott vektor x-, y- és z-irányú komponense. Végezetül a számított értékeket írassa ki az Oldalhossz.txt file-ba az alábbi módon: sngAB = érték, sngAC = érték, sngBC = érték. (4.5 pont)

2. Írja meg a „Dinamikus” nevű VBA programot, amely először a dinamikus intTomb tömbbe (integer típusú) beolvassa a „Szamok.txt” file-ban lévő „n” darab értéket („n” a file-ból beolvasott adatok száma), majd kiszámítja a long típusú lngTomb „n” darab értékét az alábbi összefüggés alapján:

$$\ln gTomb(i) = \text{int}\, Tomb(i) \times k\ ,$$

ahol k = n+1-i. (Példa: 20 elemű file esetén, ha a intTomb(1) = 5, intTomb(20) = 6, akkor lngTomb(1) = 5 × 20 = 100 és intTomb(20) = 6 × 1 = 6.) Utána számítsa ki „lngOsszeg” értékét a

$$\ln gOsszeg = \sum_{i=1}^{n} \ln gTomb(i)$$

egyenlet alapján, végezetül a MsgBox utasítással írassa ki „lngOsszeg” értékét. (5.5 pont)

6.4.1.4 4. feladatsor

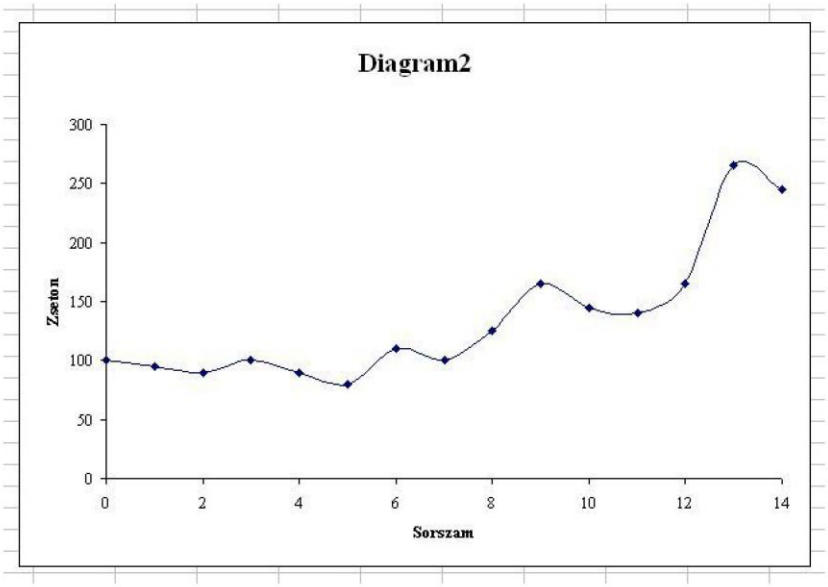
4. diger (pótdiger) – 2009. tavasz

1. Írjon a „Module 1” modulba VBA programot, amely megnyitja a „Poker-be.txt” file-t, először beolvassa kiindulási zsetonösszeget az integer típusú „intKiindulasi” paraméterbe, majd az „intSorszam()”, „intBefizetes()” és az „intNyeremeny()” dinamikus tömbökbe (integer típusúak) beolvassa a játék sorszámát (első oszlop), az adott játék során befizetett zseton mennyiségét (második oszlop), valamint a nyereményt (harmadik oszlop). A long típusú „lngOsszPenz()” dinamikus tömbbe számítsa ki az adott játék után rendelkezésre álló zsetonok összegét az alábbi képlet alapján:

lngOsszPenz(intI) = lngOsszPenz(intI-1) - intBefizetes(intI) + intNyeremeny(intI)

(intI = 1 esetén „intKiindulasi” paraméterből vonja ki/adja hozzá az első játék során befizetett/nyert zseton összegét). Ezután az „A1” cellába írja be a „Sorszám”, a „B1” cellába a „Zseton” sztringet, az „A2” cellába nulla, a „B2” cellába „intKiindulasi” értékét, majd az „A” és „B” oszlopok következő celláiba az „intSorszam()”, illetve „lngOsszPenz()” tömböki megfelelő értékeit. Utána nyissa meg írásra a „Poker-ki.txt” file-t, melybe írja bele a „Jatekszam = ”, a „Nyeremeny = ” és az „Egy jatekra eso nyeremeny = ” sztringeket, valamint a játékok számát, a nyereményt (játék végén rendelkezésre álló zsetonösszeg és „intKiindulasi” különbsége), valamint a kettő hányadosát.
Utána a program az InputBox paranccsal kérdezze meg, hogy a felhasználó meg akarja-e jeleníteni a Sorszám/Zseton adatokat. Amennyiben „i” (igen) a válasz, a program hívja meg a „Module 1”-ben lévő Diagram2() eljárást, amelyet a „Module 2”-ben lévő Diagram() eljárás „Module 1”-be történő átmásolásával, majd a megfelelő paraméterek átírásával hozzon létre. A Diagram2() eljárás bemenő paramétere a legnagyobb játék sorszáma legyen.

	A	B
1	Sorszam	Zseton
2	0	100
3	1	95
4	2	90
5	3	100
6	4	90
7	5	80
8	6	110
9	7	100
10	8	125
11	9	165
12	10	145
13	11	140
14	12	165
15	13	265
16	14	245

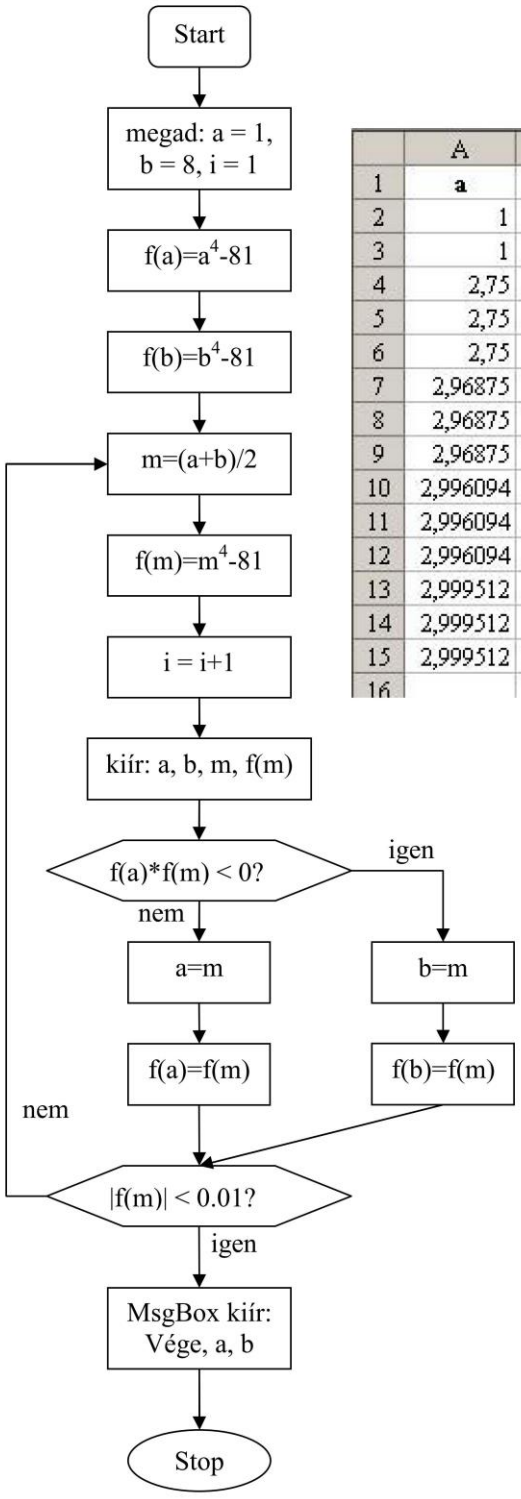


6.4.2 2009 őszi feladatsorok

6.4.2.1 1. feladatsor, A csoport

Név: **Tankör:** **Diger-1, A csoport – 2009 ősz**

1. Az alább megadott blokkdiagram alapján készítse el a „Feladat_1a” programot a Do - Loop While, vagy Do - Loop Until ciklus utasítás használatával az „$x^4 - 81 = 0$” egyenlet gyökének intervallum-felező módszerrel való meghatározására. Az „a” és „b” változók kezdőértékét InputBox segítségével adja meg. A program az „a”, „b”, „m” és „fm” értékét az alábbi ábrán látható módon írja ki (a táblázat fejléce nem programozandó). Utána írja át a program megfelelő részét úgy, hogy a a Do - Loop ciklusutasítás helyett If-Then és GoTo utasításokat alkalmazzon. A program megírása során az alábbi változókat alkalmazza: i (integer), a, b, m (mind single), fa, fb, fm (mind double). (13+2 pont).

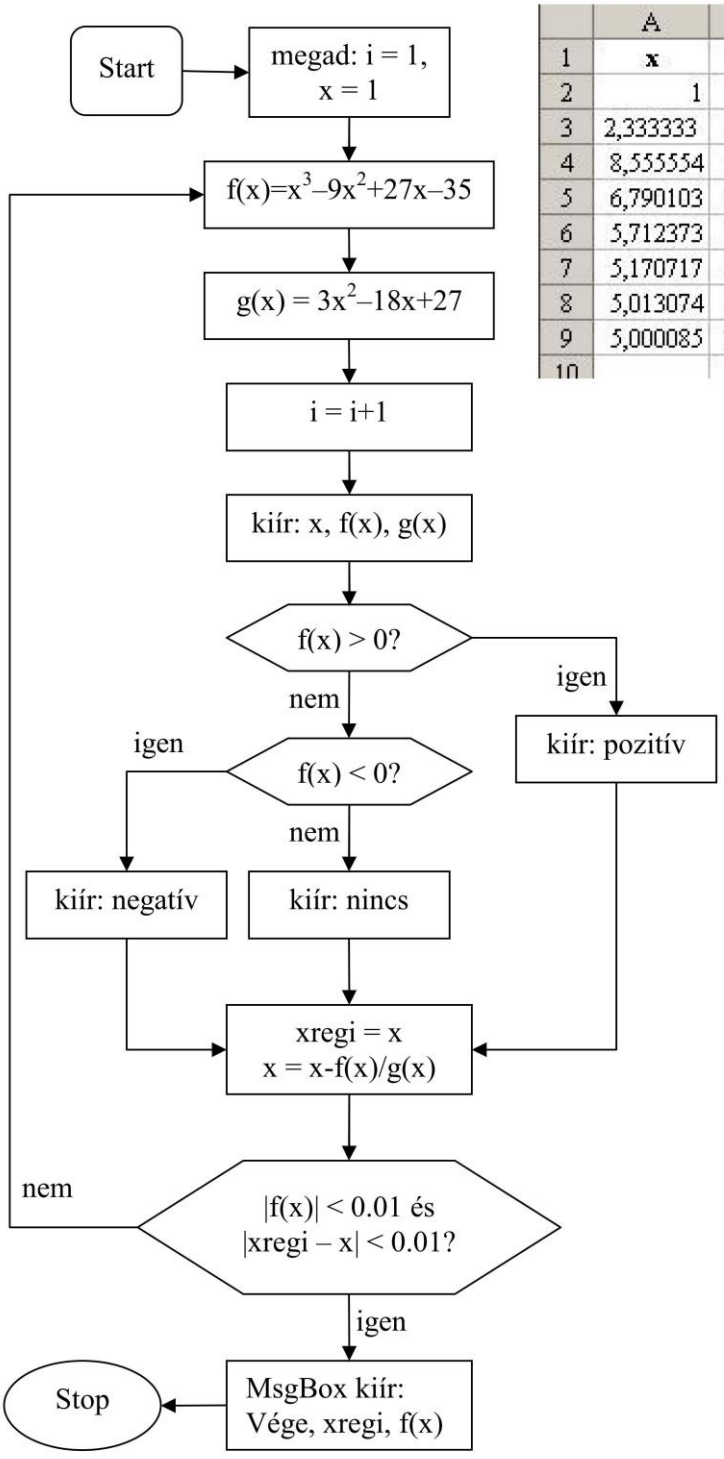


	A	B	C	D
1	**a**	**b**	**m**	**f(m)**
2	1	8	4,5	329,0625
3	1	4,5	2,75	-23,8086
4	2,75	4,5	3,625	91,67603
5	2,75	3,625	3,1875	22,22878
6	2,75	3,1875	2,96875	-3,32263
7	2,96875	3,1875	3,078125	8,772849
8	2,96875	3,078125	3,023438	2,561068
9	2,96875	3,023438	2,996094	-0,42105
10	2,996094	3,023438	3,009766	1,059849
11	2,996094	3,009766	3,00293	0,31687
12	2,996094	3,00293	2,999512	-0,05272
13	2,999512	3,00293	3,001221	0,131916
14	2,999512	3,001221	3,000366	0,039558
15	2,999512	3,000366	2,999939	-0,00659
16				

6.4.2.2 1. feladatsor, B csoport

Név: **Tankör:** **Diger-1, B csoport – 2009 ősz**

1. Az alább megadott blokkdiagram alapján készítse el a „Feladat_1b” programot a Do - Loop While, vagy Do - Loop Until ciklus utasítás használatával az „$x^3 - 9x^2 + 27x - 35 = 0$” egyenlet gyökének Newton módszerrel való meghatározására. Az „x” változó kezdőértékét InputBox segítségével adja meg. A program az „x”, „f(x)”, „g(x)” és a „hiba típusa” értékét az Excel cellákba az alábbi ábrán látható módon írja ki (a táblázat fejléce nem programozandó).
   Utána írja át a program megfelelő részét úgy, hogy a a Do - Loop ciklusutasítás helyett If-Then és GoTo utasításokat alkalmazzon. A program megírása során az alábbi változókat alkalmazza: i (integer), x, xregi (single), fx, gx (mind double). (13+2 pont).



	A	B	C	D
1	**x**	**f(x)**	**g(x)**	**Hiba**
2	1	-16	12	negatív
3	2,333333	-8,2963	1,333334	negatív
4	8,555554	163,4677	92,59255	pozitív
5	6,790103	46,44438	43,09464	pozitív
6	5,712373	11,95485	22,07091	pozitív
7	5,170717	2,228442	14,13603	pozitív
8	5,013074	0,157915	12,1574	pozitív
9	5,000085	0,001019	12,00102	pozitív
10				

6.4.2.3 2. feladatsor, A csoport

Név: **Tankör:** **Diger-2, A csoport – 2009 ősz**

1. Írja meg a „Haromszog” nevű VBA programot, amely először For-Next ciklus alkalmazásával olvassa be a „Pontok.txt” file-ból az A, B és C háromdimenziós pontok 3-3 koordinátáját (a pontok adatai a file soraiban találhatóak) az „Adatok” tömbbe (integer, méret = 3×3). Utána kiszámolja a pontokat összekötő vektorok koordinátáit az „AB”, „AC” és „BC” tömbökbe (mind integer, méret = 3), végül a „Hossz” nevű single típusú Function (3 paramétere a vektor 3 integer típusú koordinátája, és neve a vektor hosszának értékét veszi fel) segítségével számolja ki az egyes oldalak hosszúságát („ABhossz”, „AChossz” és „BChossz”, mind single típusú) az alábbi összefüggés segítségével:

$$\text{Hosszúság} = \sqrt{(x^2 + y^2 + z^2)},$$

ahol x, y és z az adott vektor három koordinátája. Végezetül a számított értékeket írassa ki az alábbi módok egyikén:
a) vagy az „Oldalhossz.txt” file-ba az alábbi módon: ABhossz = érték, AChossz = érték, BChossz = érték
b) vagy a munkalap celláiba az alábbi ábrán látható módon. Ebben az esetben a program keresse meg a leghosszabb oldalt és azt jelezze a „C” oszlop megfelelő cellájában. (12+3 pont)

Pontok.txt tartalma:
0, 0, 0 ← A pont
2, 4, 5 ← B pont
6, 4, 1 ← C pont

Oldalhossz.txt tartalma:
"ABhossz = ",6.708204
"AChossz = ",7.28011
"BChossz = ",5.656854

	A	B	C
1	ABhossz =	6,7082	
2	AChossz =	7,2801	leghosszabb
3	BChossz =	5,6569	

6.4.2.4 2. feladatsor, B csoport

Név: **Tankör:** **Diger-2, B csoport – 2009 ősz**

1. Írja meg a „Munkaber” nevű VBA programot, amely először For-Next ciklus alkalmazásával olvassa be az „Dolgozok.txt” file-ból a „Nev” tömbbe (string, méret = 5) a dolgozók nevét, a „Munka” tömbbe (long, méret = 5×3) a dolgozók órabérét, a ledolgozott órák számát és a jutalmat. Utána kiszámolja az egyes személyeknek kifizetendő bruttó összeget a „Kifizetes” tömbbe (long, méret = 5) a „Szamol” nevű long típusú Function (3 paramétere az adott munkás órabére, a ledolgozott órák száma és a jutalom) segítségével:

Kifizetés = órabér × ledolgozott órák száma + jutalom,

valamint az „Osszesen” (long) változóba kiszámolja az összesen kifizetendő bruttó munkabért.
Végezetül dolgozók nevét és a számított bruttó munkabér összegét írassa ki az alábbi módok egyikén:
a) vagy az „Osszegzes.txt” file-ba az alábbi módon: Név = kifizetendő összeg,
b) vagy a munkalap celláiba az alábbi ábrán látható módon. Ebben az esetben figyeljen rá, hogy a „B” oszlopban a fizetendő összegek ezredrésze szerepel. (12+3 pont)

Dolgozok.txt tartalma:
Pisti, 1200, 160, 8000
Jani, 1500, 120, 6000
Emese, 1300, 160, 10000
Robi, 900, 180, 15000
Ancsa, 1000, 140, 7000

Osszegzes.txt tartalma:
"Pisti = ",200000
"Jani = ",186000
"Emese = ",218000
"Robi = ",177000
"Ancsa = ",147000
"Osszesen = ",928000

	A	B	C
1	Pisti	200	eFt
2	Jani	186	eFt
3	Emese	218	eFt
4	Robi	177	eFt
5	Ancsa	147	eFt
6	Osszesen	928	eFt

6.4.2.5 3. feladatsor, A csoport

Név: **Tankör:** **Diger-3, A csoport – 2009 ősz**

1. Egészítse ki az alábbi programrészletet a jobb oldali oszlopban lévő 1-16. számú részletekkel úgy, hogy a program futtatása után eredményként a baloldali táblázatot kapjuk. (6 pont)

	A	B
1	2	1,414214
2	4	16
3	6	2,44949
4	8	64
5	10	3,162278
6	12	144
7	14	3,741657
8	16	256
9	18	4,242641
10	20	400
11	Összesen =	895,0103

```
Sub Feladat3a1()

Dim ___ Integer
Dim ___ Integer
Dim ___ Single

___ = 0

For i = 2 ___ 20 ___ 2
___ = i / 2
___ = i
If i ___ 4 = 0 Then
   Cells(j, 2) = ___
   ___
   Cells(j, 2) = ___(i)
___
___ = a + Cells(j, 2)
___ i

Cells(11, 1) = "Összesen ="
___ = a

End Sub
```

1) a
2) a
3) j
4) a As
5) i As
6) j As
7) To
8) Mod
9) Step
10) Else
11) Sqr
12) Cells(j, 1)
13) Cells(11, 2)
14) End If
15) Next
16) i ^ 2

2. Írja meg a Feladat3a2 nevű programot, amely az „Adat3a2.txt” file-ban lévő ismeretlen számú adatot (ciklusban, egyenként) beolvassa a „Be” nevű változóba (integer), majd amennyiben a „Be” érékének négyzetgyöke is egész szám, akkor „Be” értékét beírja a „Tomb” nevű dinamikus tömbbe (integer). A Tomb(0) helyen tárolja azt a számot, amely megmutatja, hogy az „Adat3a2.txt” file-ból mennyi számot írtunk át a dinamikus tömbbe. Végül a program a „Tomb” tartalmát a jobboldali táblázatban látható módon írja ki („A1” és „A2” cella tartalma is programozandó). A feladat megoldása során használja a kerekítésre szolgáló „Round” belső függvényt (pl: Round(5.6) értéke 6 lesz). (9 pont)
Amennyiben a dinamikus tömb helyett fix méretű tömböt használ, legfeljebb 6 pontot kaphat a feladatra.

	A	B
1	Adatszam =	8
2	Szamok =	9
3		81
4		169
5		196
6		64
7		36
8		144
9		121

Az „Adat3a2.txt” file tartalma (vastagon és dőlten szedve a kiválasztott számok):
5, ***9***, 7, 132, 199, 31, ***81***, ***169***, 45, ***196***, 62, ***64***, 108, 86, 39, 116, 55, ***36***, ***144***, 18, ***121***, 39, 164, 183, 24, 78

6.4.2.6 3. feladatsor, B csoport

Név: **Tankör:** **Diger-3, B csoport – 2009 ősz**

1. Egészítse ki az alábbi programrészletet a jobb oldali oszlopban lévő 1-16. számú részletekkel úgy, hogy a program futtatása után eredményként az alábbi táblázatot kapjuk. (6 pont)

	A	B	C	D	E	F	G	H	I	J	K
1	3	6	9	12	15	18	21	24	27	30	Összeg =
2	1,732051	3,320117	3	11,02318	3,872983	36,59823	4,582576	121,5104	5,196152	403,4288	594,2645

```
Sub Feladat3b1()

Dim ___ Integer
Dim ___ Integer
Dim ___ Single

___ = 0

For i = 3 ___ 30 ___ 3
___ = i / 3
___ = i
If i ___ 6 = 0 Then
   Cells(2, j) = ___(i / 5)
   ___
   Cells(2, j) = ___
___
___ = b + Cells(2, j)
___ i

Cells(1, 11) = "Összeg ="
___ = b

End Sub
```

1) b
2) b
3) j
4) b As
5) i As
6) j As
7) To
8) Mod
9) Step
10) Else
11) Exp
12) Cells(1, j)
13) Cells(2, 11)
14) End If
15) Next
16) i ^ 0.5

2. Írja meg a Feladat3b2 nevű programot, amely egy személy bevételeit és kiadásait tartalmazó „Adat3b2.txt” file-ban lévő ismeretlen számú adatot (ciklusban, egyenként) beolvassa a „Be” nevű változóba (long), majd amennyiben a „Be” abszolutértéke nagyobb, mint 10000, akkor „Be” értékét beírja a „Sok” nevű dinamikus tömbbe (long). A Sok(0) helyen tárolja azt a számot, amely megmutatja, hogy az „Adat3b2.txt” file-ból mennyi számot írtunk át a dinamikus tömbbe. Végül a program a „Sok” tömb tartalmát a jobboldali táblázatban látható módon írja ki („A1” és „A2” cella tartalma is programozandó). (9 pont)
Amennyiben a dinamikus tömb helyett fix méretű tömböt használ, legfeljebb 6 pontot kaphat a feladatra.

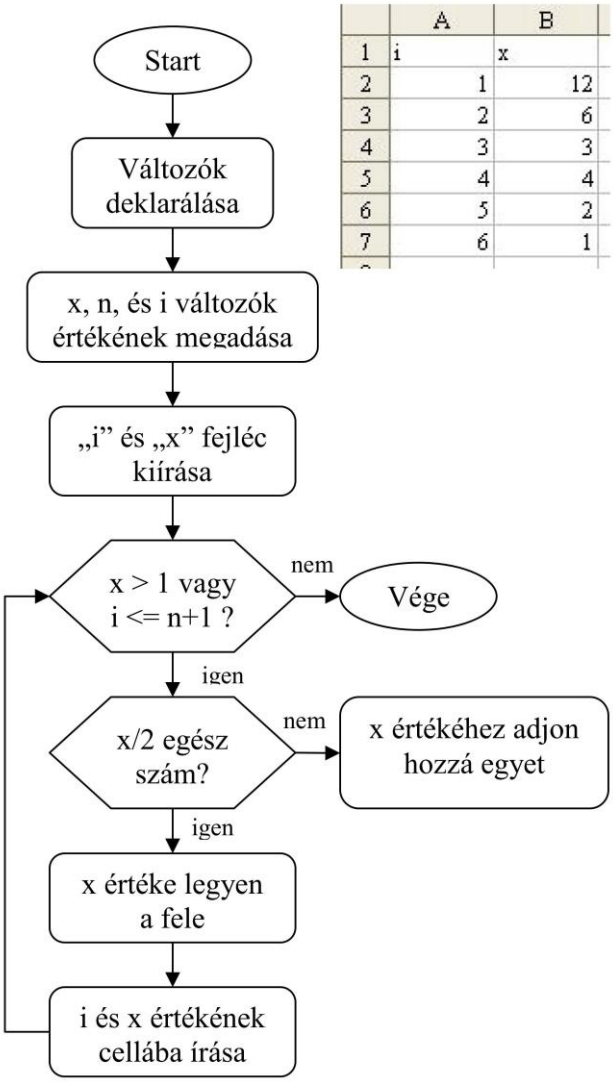
	A	B
1	Tetelszam =	4
2	Tetelek =	152396
3		-12000
4		25430
5		-20000

Az „Adat3b2.txt” file tartalma (vastagon és dőlten szedve a kiválasztott számok):
152396, ***-12000***, -6458, -3420, ***25430***, ***-20000***, 7389, -9500, -25, -679, -3100, -25, -675, 3960, 2180, -5267

6.4.2.7 4. feladatsor

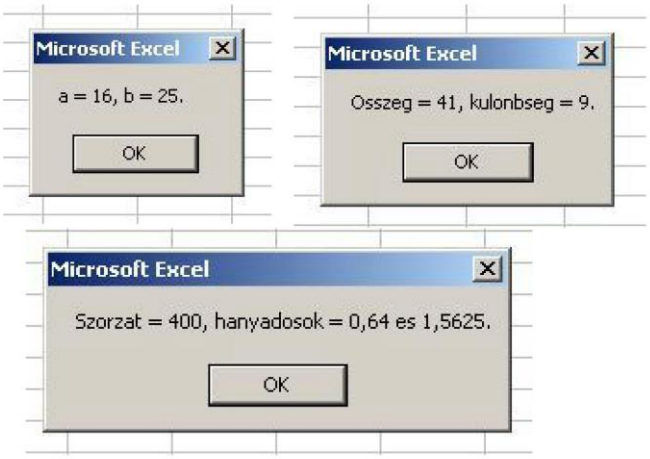
Név: **Tankör:** **Diger-4 – 2009 ősz**

1. Írja meg a Feladat1 programot a mellékelt blokkdiagram alapján úgy, hogy a program futása után eredményként (x = 11, n = 5 esetben) a jobboldali ábrát kapjuk. A program írása során az „x”, „n”, és „i” integer típusú változókat használja. A program „x” és „n” értékét InputBox paranccsal kérje be, míg „i” kezdőértéke egy legyen. A program írása során Do-While Loop ciklust alkalmazzon. (9 pont)

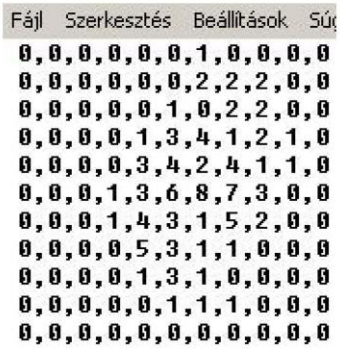


	A	B
1	i	x
2	1	12
3	2	6
4	3	3
5	4	4
6	5	2
7	6	1

2. Írja meg a Feladat2 nevű programot, amely először beolvassa „a” változó (integer) értékét az „A1”, „b” változó (integer) értékét a „B1” cellából, majd az „Osszeadas”, „Szorzas” és „Kivonas” függvények, valamint az „Osztas” szubrutin segítségével kiszámolja az a+b („Osszeg” változó, long) a×b („Szorzat” változó, long), |a-b| („Kulonbseg” változó, long), valamint a/b és b/a („Hanyados1” és („Hanyados2” változók, single) értékét. Végezetül a MsgBox parancs segítségével a változók értékeit alábbi ábrán látható módon írassa ki. (10 pont)



3. Írja meg a Feladat3 nevű programot, amely először a 11×11 méretű „Tomb” tömböt feltölti 0 értékekkel, majd „x” és „y” változóknak 6 értéket adja és a Tomb(x,y) értékét 1-re állítja. Utána 100 ciklus mindenegyes ciklusában „x” és „y” értékét -1, 0, vagy 1 értékkel változtassa meg (a -1, 0 és 1 értékeket az „Int(3 * Rnd) - 1” kifejezéssel állíthatja elő). Amennyiben „x” vagy „y” értéke kisebb lesz, mint 1, akkor a változó értékéhez adjon hozzá 11-t, míg ha „x” vagy „y” értéke nagyobb lesz, mint 11, akkor a változó értékéből vonjon ki 11-t, majd a Tomb(x,y) értékét növelje meg 1-vel. A 100 ciklus után a „Tomb” tartalmát a jobboldali ábrán látható módon írja ki az Eredmeny.txt file-ba. (11 pont)



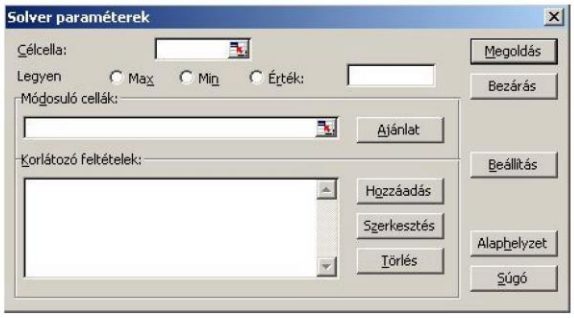
6.4.3 2010 őszi feladatsorok

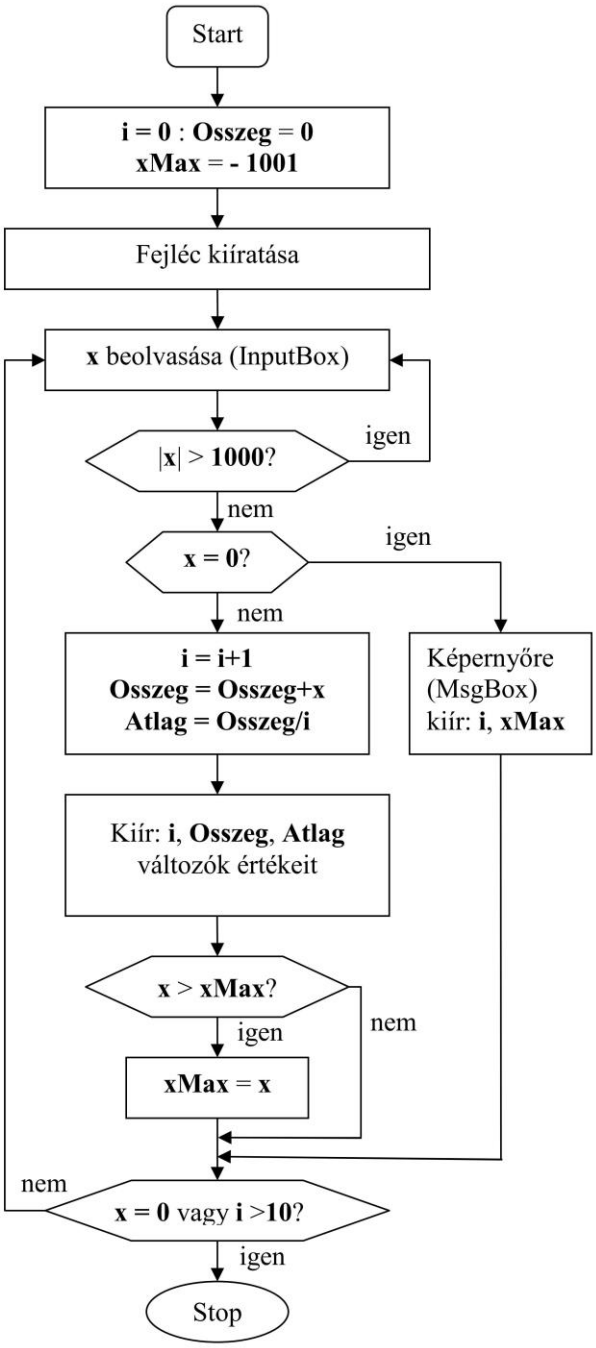
6.4.3.1 1. feladatsor, A csoport

Név: **Tankör:** **Diger-1, A csoport – 2010 ősz**

1. Egy téglatest élei: „a”, „b” és „c”. Az élek hosszáról tudjuk, hogy a + b + c = 12 egység és b = a / 2 + 2. A Solver segítségével keressük azt a téglatestet, amelynek a térfogata a legnagyobb. Adja meg, hogy milyen képletet kell írnunk a B3, B4 és B5 cellákba, melyik legyen a célcella, mire állítsuk be a „Legyen” értékét. (lásd a mellékelt ábrák) (2 pont)

	A	B
1	"a" él=	2
2	"b" él=	3
3	"c" él=	7
4	Térfogat=	42





2. A baloldalt megadott blokkdiagram alapján készítse el a „Beolvasas” programot a Do - Loop While ciklus utasítás használatával. Az „**x**” változó kezdőértékét InputBox segítségével adja meg. A program a táblázat fejlécét, valamint az „**i**”, „**Osszeg**” (a beadott „**x**” értékek összege) és „**Atlag**” (a beadott „**x**” értékek átlaga, **Osszeg** / **i**) értékét az alábbi ábrán látható módon Excel cellákba írja ki, míg „**i**” és „**xMax**” értékét a MsgBox paranccsal a képernyőre írja ki. A program megírása során használt változók közül **i, x** és **xMax** integer, **Osszeg** long és **Atlag** single legyen. Milyen **xMax** értéket ír ki a program, ha a beolvasott x értékek (az alábbi ábrán megadott kiiratásnak megfelelően) rendre 5, 9, -3, 42, -35, 6 és 0? (13 pont).

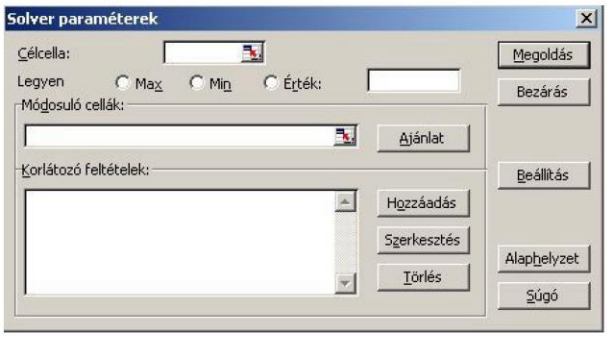
	A	B	C
9			
10	i	Osszeg	Atlag
11	1	5	5
12	2	14	7
13	3	11	3,66667
14	4	53	13,25
15	5	18	3,6
16	6	24	4

6.4.3.2 1. feladatsor, B csoport

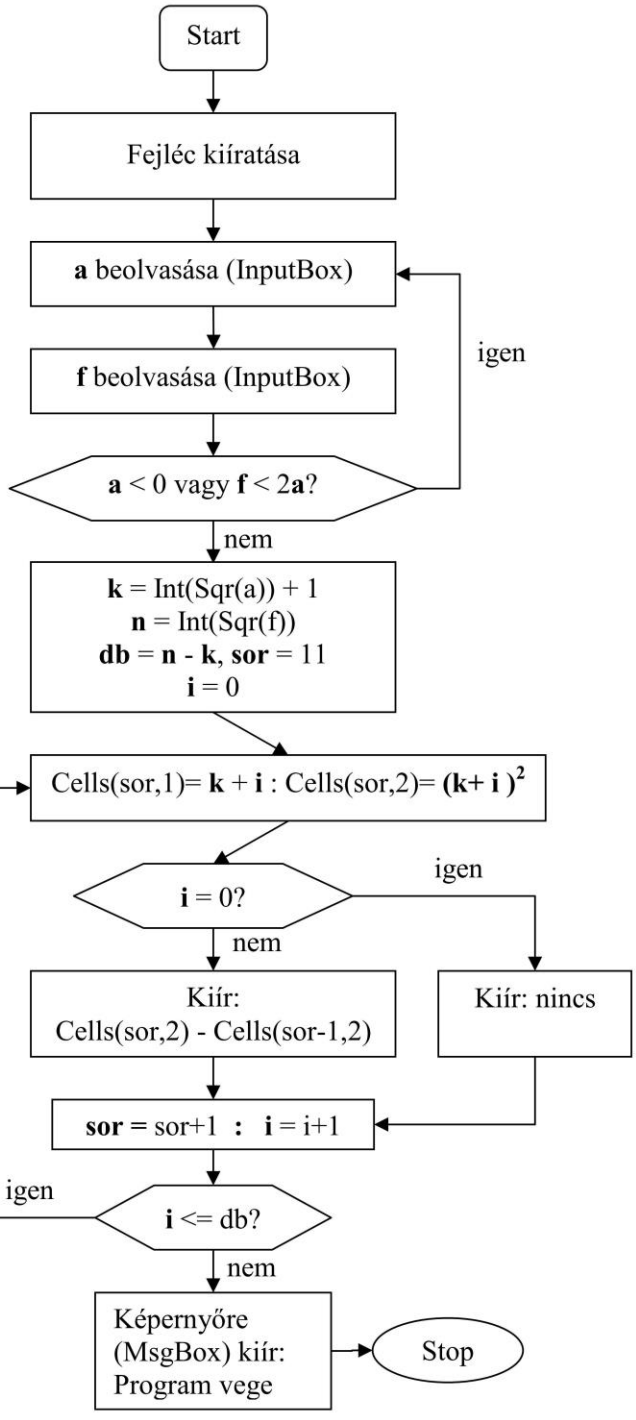
Név: **Tankör:** **Diger-1, B csoport – 2010 ősz**

1. Az $y = a * x^2 + b * x + c$ másodfokú egyenlet „a”, „b” és „c” együtthatóiról tudjuk, hogy a + b + c = 9 és a=b/4. A Solver segítségével azokat az együtthatókat keressük, melyeknél az egyenlet $D=b^2-4ac$ diszkriminánsa zérus. Adja meg, hogy milyen képletet kell írnunk a B2, B4 és B5 cellákba, melyik legyen a célcella, mire állítottuk be a „Legyen” értékét. (lásd a mellékelt ábrák) (2 pont)

	A	B
1		
2	a=	0,5
3	b=	2
4	c=	6,5
5	D=	-9



2. Az jobboldalt megadott blokkdiagram alapján készítse el a „Negyzetek” programot a Do - Loop While ciklus utasítás használatával. Az „**a**” és „**f**” változók kezdőértékét InputBox segítségével adja meg. Utána a program a táblázat fejlécét, valamint a „**k** + **i**”, „$(\mathbf{k} + \mathbf{i})^2$” és „$(\mathbf{k} + \mathbf{i})^2 - (\mathbf{k} + \mathbf{i} - 1)^2$” értékeket az alábbi ábrán látható módon (**a** = 300, **f** = 600) Excel cellákba írja ki, míg a „Program vege” üzenetet a MsgBox paranccsal a képernyőre írja ki. Az Int belső függvény paraméterének egész részét számolja, pl. Int(5.6) értéke 5. A program megírása során használt változók mind integer típusúak legyenek. (13 pont).



	A	B	C
9			
10	Szam	Negyzet	Kulonbseg
11	18	324	nincs
12	19	361	37
13	20	400	39
14	21	441	41
15	22	484	43
16	23	529	45
17	24	576	47

6.4.3.3 2. feladatsor, A csoport

Név: **Tankör:** **Diger-2, A csoport – 2010 ősz**

1. Írja meg a **Megoldas1** nevű VBA programot, az adott $f(k)=3/(Ak^2+Bk+C)$ függvény aktuális értékeinek, és a $Sum = \sum_{k=1}^{m} f(k) = f(1) + f(2) + ... + f(m)$ összeg kiszámítására.

A másodfokú polinom A, B, C együtthatóit rendre a jobb oldalon látható *egyutthatok.txt* tartalmazza. Az együtthatókat az ADAT nevű kétindexes (5 x 3) méretű tömbbe olvastassa be, és írja ki az Excel cellákba. A függvény értékek és az egyes függvényekhez tartozó SUM értékek tárolására ERTEK kétindexes dinamikus (5 x m), ill. SUM egyindexes (5) tömböket deklaráljon. Az **m** értékét Inputbox-szal olvastassa be (legyen 4).
A függvényértékek számításához deklaráljon F nevű, single értékű Function-t, melynek 4 paramétere, k, A, B és C Integer típusúak.
A SUM tömb elemeit írja ki *osszegek.txt* fájlba.
Az Ertek tömb elemeit írassa ki az Excel munkalapra az alábbi, *m=4* értéknek megfelelő ábrán látható módon. **(Összesen: 15 pont)**

Együtthatok.txt

A	B	C
1	5	6
2	3	4
3	3	-2
2	4	-1
4	4	4

	A	B	C	D	E	F	G	H	I
1	**A**	**B**	**C**		**k =**	**1**	**2**	**3**	**4**
2	1	5	6			0,25	0,15	0,10	0,07
3	2	3	4			0,33	0,17	0,10	0,06
4	3	3	-2			0,75	0,19	0,09	0,05
5	2	4	-1			0,60	0,20	0,10	0,06
6	4	4	4			0,25	0,11	0,06	0,04

6.4.3.4 2. feladatsor, B csoport

Név: **Tankör:** **Diger-2, B csoport – 2010 ősz**

1. Írja meg a **D2_B** nevű VBA programot, amely először a *Koordinatak.txt* file-ban lévő egész számokat (lásd baloldali ábra, az egyes sorokban levő számpárok rendre egy síkbeli pont (x,y) koordinátái), beolvassa az **X** ill. **Y** dinamikus tömbökbe, a **db** változóban tárolja a beolvasott koordináta-párok darabszámát, majd definiálja a kétindexes **db** × **db** méretű **Matrix** nevű Double típusú dinamikus tömböt. A **Matrix** tömb elemeinek az értékét az alábbi módon számítsa ki:

$$Matrix(i,j) = \sqrt{(X(i) - X(j))^2 + (Y(i) - Y(j))^2}$$

Minden egyes **Matrix(i,j)** tömbelem értékét írja ki egy Excel munkalapra, az (i, j) indexeknek (sor, oszlop) szerint megfelelő cellába. A **Matrix** tömb elemeit írassa ki a *kimenet.txt* fájlba is.

Változtassa meg a **D2_B** programot úgy, hogy a **Matrix(i,j)** tömbelemek értékét a **Tavolsag** Function hívásával számítsa. Készítsen **Tavolsag** nevű, Double típusú Function-t, melynek négy formális paramétere (**a1, a2, b1, b2)** legyenek Integer típusúak (Nem szükséges az egész **D2_B** programot újra leírnia, csak a változtatandó részt.). **(Összesen: 15 pont)**

	A	B	C	D	E	F	G	H
1	X0	Y0		Számolás az alap Subrutinban				
2	0	0		0,00	5,00	10,63	13,00	
3	3	4		5,00	0,00	5,83	9,06	
4	8	7		10,63	5,83	0,00	4,47	
5	12	5		13,00	9,06	4,47	0,00	
6								
7								

6.4.3.5 3. feladatsor, A csoport

Név: **Tankör:** **Diger-3, A csoport – 2010 ősz**

1. Egészítse ki az alábbi programrészletet a jobb oldali oszlopban lévő 1-16. számú részletekkel úgy, hogy a program futtatása után eredményként a jobboldali táblázatot kapjuk. (4 pont)

```
Sub Program_D3_1A()

Dim ___
Dim j As Integer
Dim ___ Integer
Dim ___ Single

___ i = 1 ___ 15
Cells(___) = i
Cells(1, i + 1) = i
For j = i To 15
    k = ___
    Cells(___) = k
    m = Sqr(___)
    ___ m = Int(m) Then
        Cells(___) = m
        ___
        Cells(i + 1, j + 1) = ___
    ___
Next ___
Next ___

End Sub
```

1) k
2) For
3) j
4) i
5) i%
6) "-"
7) m As
8) k As
9) i ^ 2 + j ^ 2
10) If
11) To
12) End If
13) Else
14) i + 1, j + 1
15) i + 1, 1
16) j + 1, i + 1

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	2	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4	3	10	13	-	5	-	-	-	-	-	-	-	-	-	-	-
5	4	17	20	25	-	-	-	-	-	-	-	-	-	-	-	-
6	5	26	29	34	41	-	-	-	-	-	-	-	13	-	-	-
7	6	37	40	45	52	61	-	-	10	-	-	-	-	-	-	-
8	7	50	53	58	65	74	85	-	-	-	-	-	-	-	-	-
9	8	65	68	73	80	89	100	113	-	-	-	-	-	-	-	17
10	9	82	85	90	97	106	117	130	145	-	-	-	15	-	-	-
11	10	101	104	109	116	125	136	149	164	181	-	-	-	-	-	-
12	11	122	125	130	137	146	157	170	185	202	221	-	-	-	-	-
13	12	145	148	153	160	169	180	193	208	225	244	265	-	-	-	-
14	13	170	173	178	185	194	205	218	233	250	269	290	313	-	-	-
15	14	197	200	205	212	221	232	245	260	277	296	317	340	365	-	-
16	15	226	229	234	241	250	261	274	289	306	325	346	369	394	421	-

Adja meg, hogy a C3-as cellában milyen számértéket írt felül a benne jelenleg látható "-" karakter! (1 pont)

Fájl Szerkesztés

```
Andras,5
Balazs,4
Cecilia,2
Dezso,1
Elemer,3
Ferenc,3
Gabor,3
Hedvig,2
Ilona,4
Judit,1
Kalman,5
Laszlo,2
Melinda,3
Nora,1
Oszkar,3
Peter,4
Ramona,3
Sandor,3
Tibor,2
Ubul,3
Xaver,4
Yvonne,4
Vajk,4
Zoltan,2
```

	A	B	C	D	E
20				Jegy	Db
21	Andra	5		1	3
22	Balaz	4		2	5
23	Cecil	2		3	8
24	Dezs	1		4	6
25	Elem	3		5	2
26	Ferer	3			
27	Gabo	3			
28	Hedvi	2			
29	Ilona	4			
30	Judit	1			
31	Kalm	5			
32	Laszl	2			
33	Melin	3			
34	Nora	1			
35	Oszk	3			
36	Peter	4			
37	Ramo	3			
38	Sand	3			
39	Tibor	2			
40	Ubul	3			
41	Xaver	4			
42	Yvon	4			
43	Vajk	4			
44	Zolta	2			

2. Írja meg a **D3_2A** nevű programot, amely:
- a baloldali, bekeretezett ábrán látható „Adatok.txt” fájlban lévő ismeretlen számú név-jegy adatpárt (ciklusban, egyenként) beolvassa a **nev** nevű (String típusú) és **jegy** nevű (Integer típusú) változókba,
- a fejlécet, valamint a **nev** és **jegy** változók értékét az ábrán látható módon kiírja egy Excel munkalap celláiba,
- beolvasás közben a **Jegyek** nevű (Integer típusú), egyindexes tömbben tárolja a különböző osztályzatok darabszámát, amelyeket végül az ábrán látható módon kiír, és MsgBox használatával kiírja a jegyek átlagát is.

A program minden változóját deklarálja (típusuk megadásával)!

(10 pont)

6.4.4 2011 őszi feladatsorok

6.4.4.1 1. feladatsor, A csoport

Név: **Tankör:** **Diger-1, A csoport – 2011 ősz**

1. A megadott blokkdiagram alapján készítse el a „A1a” programot az if - then és a goto parancsok használatával. Az „**A**” változó kezdőértékét InputBox segítségével adja meg. Figyeljen arra, hogy az Inputbox paramétereit úgy adja meg, hogy az ábrán látható ablakot eredményezze. A program a táblázat fejlécét, valamint az „**A**”, „**N**” értékét az alábbi ábrán látható módon Excel cellákba írja ki, míg „**Program vege**” szöveget a MsgBox paranccsal a képernyőre írja ki. A program megírása során használt változók integer típusúak legyenek. A maradékosztáshoz használja a „**Mod**” parancsot. (11 pont)
Írja át a programot úgy, hogy a ciklust létrehozó if - then és a goto parancsok helyett, Do - Loop While ciklusutasítást, az if - then - else szerkezet helyett pedig a Select Case szerkezetet használja. (4 pont)

	A	B	C
1			
2	No.	Szam	
3	1	70	
4	2	72	
5	3	24	
6	4	8	
7	5	9	
8	6	3	
9	7	1	
10			

6.4.4.2 1. feladatsor, B csoport

Név: **Tankör:** **Diger-1, B csoport – 2011 ősz**

1. A megadott blokkdiagram alapján készítse el a „B1a” programot a For - Next és a Do - Loop While ciklusutasítások használatával. Az „**Max**” változó kezdőértékét InputBox segítségével adja meg. Figyeljen arra, hogy az Inputbox paramétereit úgy adja meg, hogy az ábrán látható ablakot eredményezze. A program a táblázat fejlécét, valamint a 3. sor celláinak értékét az alábbi ábrán látható módon írja ki, míg „**Program vege**” szöveget a MsgBox paranccsal a képernyőre írja ki. A program megírása során használt változók integer típusúak legyenek. A véletlenszerű 0, 1, és 2 értékek létrehozásához a „3 * Rnd” és az egészosztás parancsokat használja. (13 pont)
   Írja át a programot, az if - then - else szerkezet helyett a Select Case szerkezetet használja! (2 pont)

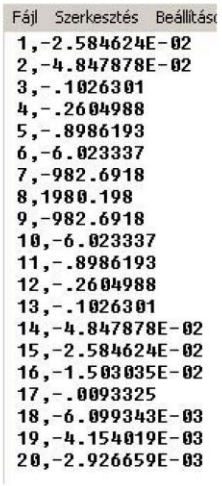
	A	B	C
1			
2	A>B	A=B	A<B
3	24	15	21

6.4.4.3 2. feladatsor, A csoport

Név: **Tankör:** **Diger-2, A csoport – 2011 ősz**

1. Írja meg az „A1” programot, amely először az Inputbox utasítás segítségével beolvassa „n” változó értékét (figyeljen arra, hogy az Inputbox paramétereit úgy adja meg, hogy az az ábrán látható ablakot eredményezze!), majd definiálja a „Tomb” nevű „n”-méretű dinamikus tömböt. Utána „i” = 1-től egyesével „n”-ig haladva számolja ki a „Tomb” változó „i.” eleméhez a megfelelő $2 \cdot f(i) - (f(i-1) + f(i+1))$ értéket, ahol f(i) értékét a $\frac{10}{(i-8)^2 + 0{,}01}$ kifejezés adja meg. A kifejezés kiszámolására írja meg az „f” nevű függvényt. A program határozza meg „Tomb” legnagyobb értékét („Max” változó), „i”-t, „Tomb” értékeit és a legnagyobb értéket az ábrán látható módon írja ki a munkalapra (a fejléc is programozandó), valamint a felhasználó döntése esetén (Inputbox) „i” és „Tomb” értékeit az ábrán látható módon a „Tomb.txt” file-ba írja ki. A főprogram megírása során az „i”, „n”, „Max”, „Tomb” és „Dontes” (utóbbi string) változókat használja, „f” függvény megírása során (amennyiben szükséges) további változókat is alkalmazhat. (15 pont)

	A	B	C	D	E
1	i	Ertek		Max =	1980,198
2	1	-0,02585			
3	2	-0,04848			
4	3	-0,10263			
5	4	-0,2605			
6	5	-0,89862			
7	6	-6,02334			
8	7	-982,692			
9	8	1980,198			
10	9	-982,692			
11	10	-6,02334			
12	11	-0,89862			
13	12	-0,2605			
14	13	-0,10263			
15	14	-0,04848			
16	15	-0,02585			
17	16	-0,01503			
18	17	-0,00933			
19	18	-0,0061			
20	19	-0,00415			
21	20	-0,00293			
22					



6.4.4.4 2. feladatsor, B csoport

Név: **Tankör:** **Diger-2, B csoport – 2011 ősz**

1. Írja meg az „B1” programot, amely először az Inputbox utasítás segítségével beolvassa „n” változó értékét (figyeljen arra, hogy az Inputbox paramétereit úgy adja meg, hogy az az ábrán látható ablakot eredményezze!), majd definiálja a „Matrix” nevű „n × n”-méretű dinamikus tömböt. Utána a „Matrix” minden egyes (i, j) eleméhez (i: sorindex, j: oszlopindex) számolja ki a $g(i,i)+g(i,j)-g(j,i)+g(j,j)$ értéket, ahol $g(x,y)=x^2+2\cdot y$. A g(x,y) kifejezés kiszámolására írja meg a „g” nevű függvényt. A program a határozza meg „Matrix” elemeinek összegét („Osszeg” változó), valamint „Matrix” értékeit a munkalapra, míg „Osszeg” értékét a „Matrix.txt” file-ba írja ki az ábrákon látható módon. (12 pont)
   A programot írja át úgy, hogy a „g” függvény helyett a „g2” szubrutint alkalmazza, mely a $g(i,i)+g(i,j)-g(j,i)+g(j,j)$ értékét a kifejezés tagjainak összevonása után kapott $2\cdot i^2+4\cdot j$ összefüggéssel határozza meg. (3 pont)
   A főprogram megírása során az „i”, „j”, „n”, „Matrix” és „Osszeg” változókat használja, „g” függvény és „g2” szubrutin megírása során (amennyiben szükséges) további változókat is alkalmazhat.

	A	B	C	D	E	F	G	H
1	6	10	14	18	22	26	30	34
2	12	16	20	24	28	32	36	40
3	22	26	30	34	38	42	46	50
4	36	40	44	48	52	56	60	64
5	54	58	62	66	70	74	78	82
6	76	80	84	88	92	96	100	104
7	102	106	110	114	118	122	126	130
8	132	136	140	144	148	152	156	160

6.4.4.5 3. feladatsor, A csoport

Név: **Tankör:** **Diger-3, A csoport – 2011 ősz**

1. Írja meg az „A1” programot, amely először az Adatok.txt file tartalmát (pl.: 1, 20, 36, 15, 96, 48, 55, 72, 28, 88, 92, 35, 14, 19, 62, 49, 76, 32, 99, 44) olvassa be a „Tomb” nevű dinamikus tömbbe (használja az EOF kifejezést!). Utána az Inputbox utasítások segítségével olvasson be alsó és felső határértéket az „also” és a „felso” változókba (figyeljen arra, hogy az Inputbox paramétereit úgy adja meg, hogy az az ábrán látható ablakot eredményezze!). Ezután a program keresse ki a „Tomb” dinamikus tömb elemei közül azokat, amelyekre igaz, hogy also <= Tomb(i) <= felso, összesítse, hány megfelelő számot talált, majd az eredményt az alábbi ábrán látható módon írja ki a munkalapra. Amennyiben a program nem talál a „Tomb” dinamikus tömb elemei közül feltételeknek megfelelőt, ezt MsgBox ablakkal is hozza a felhasználó tudomására! A felsorolt változók legyenek integer típusúak! A feladat megoldása során (amennyiben szükséges) további változókat is alkalmazhat (pl.: i, j, k, talalat, stb.) (15 pont)

	B	C	D	E
1	Ertek		Also hata	50
2			Felso hata	54
3			Talalat =	0
4				
5				

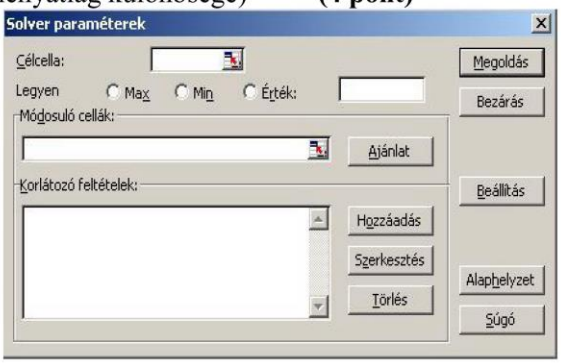
	A	B	C	D	E
1	No.	Ertek		Also hatar =	15
2	1	20		Felso hatar =	26
3	2	15		Talalat =	3
4	3	19			
5					

6.4.4.6 4. feladatsor, A csoport

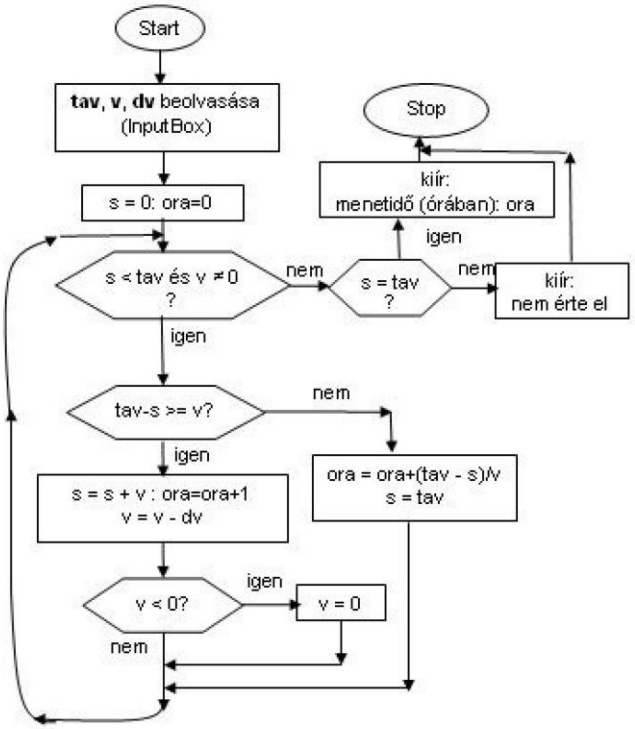
Név: **Tankör:** **Diger-4, A csoport – 2011 ősz**

1. A 2010. 09. 28-i Zalaegerszeg-Videoton labdarúgó-mérkőzés eredményére fogadtunk úgy, hogy az elérhető fogadóirodákból kiválasztottuk azt a hármat, amely a mérkőzés adott kimenetele (1 ; x ill. 2) esetén a tétből a legnagyobb szorzóval (3,20 ; 3,59 ill. 2,40) számított nyereményt fizeti. Összesen 10000 forintot tettünk meg a három kiválasztott fogadóirodánál oly módon, hogy célunkhoz – függetlenül a mérkőzés végeredményétől ugyanakkora nyereményt kívántunk elérni – az összesen 10000 Ft-nyi tét felosztását Solver-rel számítottuk. Milyen képleteket használtunk a B5, D3-D5, D7, F3-F5 és F7 cellák értékeinek kiszámolásához? Melyek voltak a módosuló cellák, a célcella, illetve mire állítottuk be a „Legyen” értékét? (A hiba az adott nyeremény és a nyereményátlag különbsége) **(4 pont)**

	A	B	C	D	E	F
1						
2		Tét	Szorzó	Nyeremény		Hibanégyzet
3	Zalaegerszeg		3,20			
4	Döntetlen		3,59			
5	Videoton		2,40			
6						
7			Átlag =		Összesen =	



2. Írjon VBA programot a mellékelt blokkdiagram alapján. A programban használjon Do - Loop ciklusutasítást, a **tav**, **v** és **dv** változók kezdőértékét InputBox segítségével adja meg, és minden változót deklaráljon Single-ként. **(8 pont)**



3. Készítsen VBA **Function**-t, melynek neve legyen **fajlagos,** formális paraméterei **r** és **m** (egy henger alaplapjának sugara és a henger magassága) **Double** típusúak. A **Function** neve vegye fel a henger **A/V** fajlagos felületének (**Double** típusú) értékét, ahol $\mathbf{A}= 2\mathbf{r}^2\pi+2\mathbf{r}\pi\mathbf{m}$ és $\mathbf{V}=\mathbf{r}^2\pi\mathbf{m}$. A **Function**-ban deklaráljon **pi** nevű, **Double** típusú változót π értékének tárolására: **pi** = 4 * atn (1)
A főprogramban olvastassa be Inputbox segítségével egy For-Next cikluson belül 10 henger **r** és **m** adatait, és számítsa a **fajlagos Function** hívásával a megfelelő fajlagos felületeket. Az **r**, **m** és fajlagos felület értékeket irassa ki egy Excel munkalap első 10 sorának A, B és C oszlopaiba. **(6 pont)**

4. Készítsen VBA programot, amely először megnyitja az „Adatok.txt” fájlt, majd a benne lévő adatpárokat beolvassa az (1. oszlop) **x Integer** típusú illetve az (2. oszlop) **y Single** típusú egyindexes dinamikus tömbökbe. Az **n** (**Integer** típusú) változóban tárolja a beolvasott adatok számát, amely egyben az **x** tömb legnagyobb értékű eleme. Az **a** és **b** változókba InputBox-szal olvastasson be két egész számot, melyekre vizsgálja $\mathbf{0 \le a < b \le n}$ feltételek teljesülését. (Ha nem teljesülnek, a beolvastatást a program „**a** nem jó” ill. „**b** nem jó” üzenetet adva, a feltételek teljesüléséig ismételje.) Ezután a program számítsa ki az $S = \sum_{k=a+1}^{b} \frac{y(k)+y(k-1)}{2}$ összeget, majd írja ki **a**, **b** és **S** értékét. A program minden változóját deklarálja! **(12 pont)**

Fájl	Szerkeszté
0,	9
1,	7.5
2,	6
3,	4.5
4,	3
5,	1.5
6,	2.25
7,	3
8,	3.75
9,	4.5
10,	5.25
11,	6

6.4.5 2012 őszi feladatsorok

6.4.5.1 1. feladatsor, A csoport

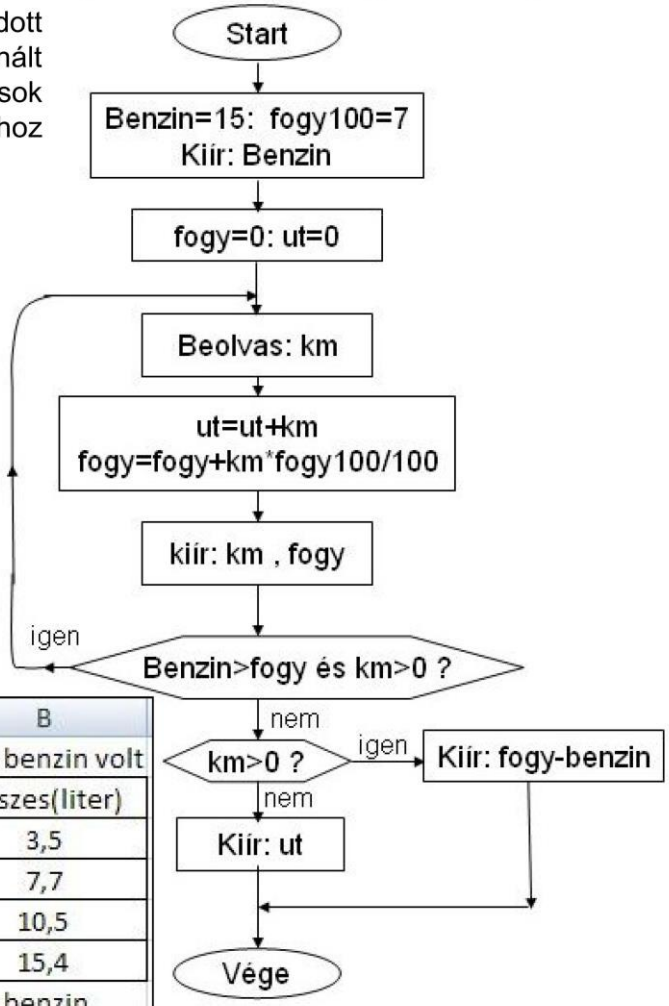
A	**Z1 (2012. okt.30.)**	Név: tk:

1. Írja be az alább (jobbra) látható Excel „Munkalap” megfelelő celláiba, hogy mit ír ki az alább (balra) megadott VBA program. (4,5 p.)

```
Sub F1_A()
Dim n As Integer, k%, b#
    n = (3 ^ 2 + 10) / 3: k = 2
vissza:
    b = (k ^ 2 + 1) / 2
    Cells(k, 1) = k: Cells(k, 2) = b
    k = k + 1
  If k < n Then GoTo vissza
  Cells(1, 1) = "k": Cells(1, 2) = "b"
  Cells(k, 1) = "n=": Cells(k, 2) = n
End Sub
```

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

2. Készítsen VBA programot az itt megadott blokkdiagram alapján. A programban használt összes változót deklarálja, a beolvasások InputBox-szal történjenek, a „hurok” leírásához hátul tesztelt ***Do - Loop*** ciklust használjon.



A programban a kiiratások (az oszlopok fejlécét és az egyéb szöveges részeket is beleértve) az alábbi ábrák szerint történjenek. Az egyes ábrákhoz tartozó beolvasott ***km*** értékek mindkét ábrán az ***A*** oszlopban láthatók. A baloldali ábra olyan kiiratást mutat, amikor a ***km*** utolsó beolvasott ***0*** értéke okozza a ciklusból való kilépést. A jobboldali ábra a ciklusból való olyan kilépést mutat, amikor a ***km*** utolsó beolvasott értékéhez tartozó benzin szükséglet már nem áll rendelkezésre. (10,5 p.)

	A	B
1	15	liter benzin volt
2	Rész(km)	Összes(liter)
3	50	3,5
4	60	7,7
5	40	10,5
6	0	10,5
7	150	km utat tettél
8		Megérkeztél!

	A	B
1	15	liter benzin volt
2	Rész(km)	Összes(liter)
3	50	3,5
4	60	7,7
5	40	10,5
6	70	15,4
7	0,4	liter benzin
8	kellene még a célhoz	

6.4.5.2 1. feladatsor, B csoport

Név:	tk:	Z1 (2012. okt.30.)	B

1. Írja be az alább (jobbra) látható Excel „Munkalap” megfelelő celláiba, hogy mit ír ki az alább (balra) megadott VBA program. (4,5 p.)

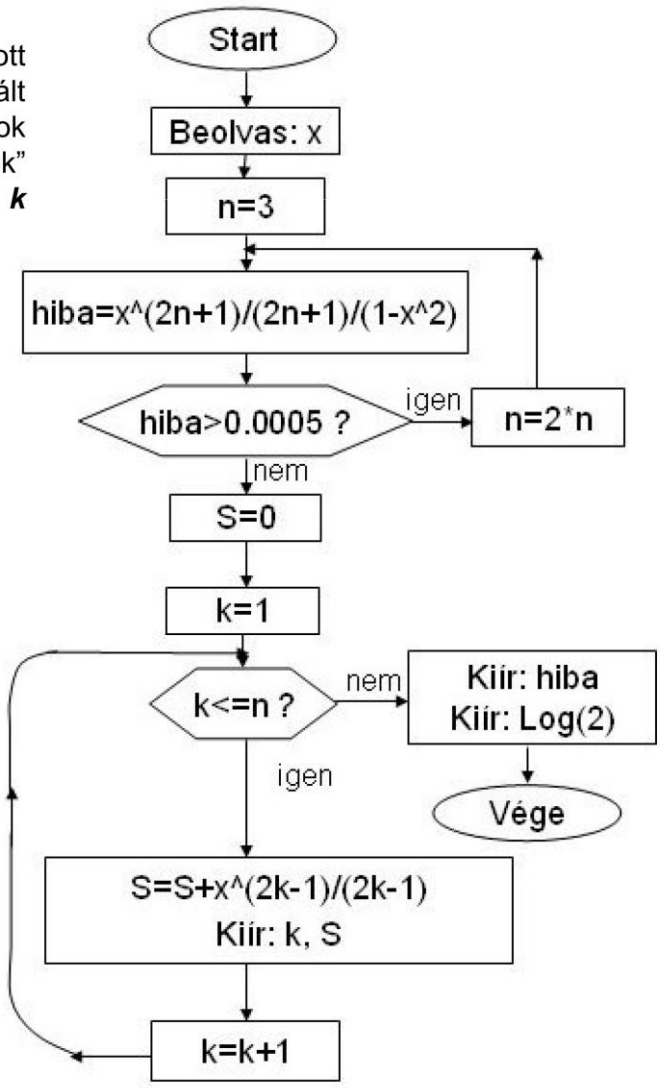
```
Sub F1_B()
Dim x As Double, y#, m%
   x = 1 + 3 * 0.2 ^ 2: m = 5
   Cells(1, 1) = "x=": Cells(1, 2) = x
ujra:
     m = m - 1
     y = 3 * (m + 1) / 2
     Cells(2, m) = m: Cells(3, m) = y
  If m - x > 1 Then GoTo ujra
   m = m - 1
 Cells(2, m) = "m": Cells(3, m) = "y"
End Sub
```

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

2. Készítsen VBA programot az itt megadott blokkdiagram alapján. A programban használt összes változót deklarálja, a beolvasások InputBox-szal történjenek, a (második) „hurok” leírásához elől tesztelt ***Do – Loop,*** (vagy ***k*** ciklusváltozójú ***For – Next***) ciklust használjon.

A programban a kiiratások (az oszlopok fejlécét és az egyéb szöveges részeket is beleértve) az alábbi ábra szerint történjenek. (Az ábra az x=0,6 input értékhez tartozó kiiratásokat mutatja). (10,5 p.)

	A	B
1	k	S
2	1	0,6
3	2	0,672
4	3	0,6876
5	4	0,6916
6	5	0,6927
7	6	0,6930
8		
9	hiba=	1,57E-04
10	ln2=	0,6931



6.4.5.3 2. feladatsor, A csoport

A	Z1 (2012. nov.29.)	Név:	tk:

A jobbra látható ***IskTam.txt*** fájl az ***Iskolák*** szöveg után a támogatott iskolák számát, nevét és az adományokból való részesedési hányadát tartalmazza, majd a ***támogatók*** szöveget a támogatók száma, neveik, és az egyes nevek mellett az első ill. második félévi adomány ezer Ft-ban megadott összegei követik. Az alább megadott feladatokat úgy oldja meg, hogy a program olyan (az ***IskTam.txt*** fájllal azonos szerkezetű) input fájl esetén is helyes eredményt adjon, ahol az iskolák és a támogatók száma más, de a támogatók száma <= 100.

```
Iskolák, 3
Focisuli, Képző, Alap
0.4, 0.25, 0.35
támogatók, 10
Balogh, 22,  0
Cecey,  15,  0
Daróci,  0, 14
Fodor,  25,  0
Kiss,    0, 35
Nagy,   15, 13
Papp,    0, 24
Szabó,   0, 43
Tóth,   32,  0
Zalai,  15,  0
```

1. Készítsen ***Tamogat*** nevű VBA programot a fájl adatainak beolvasására, és az adatokból az egész évi összes támogatás, valamint ezen összegből az egyes iskolákra jutó részek kiiratására az alább látható táblázat ***A, B, C*** oszlopa szerinti formában.

Az iskolák nevét egy ***String*** típusú, ***Isk*** nevű tömbbe, az egyes iskoláknak jutó támogatási hányad értékeket egy ***Double*** típusú, ***resz*** nevű tömbbe olvastassa. Ez a két tömb dinamikus indexhatárú legyen. A k-adik támogató első ill. második féléves adományait az ***eFt%(100,2)*** –ként deklarált tömb ***(k,1)*** ill ***(k,2)*** indexeknek megfelelő elemébe olvastassa. A programban használt összes változót deklarálja. **(7,5 pont)**

(Ha az ***Isk*** és ***resz*** tömböket nem dinamikus indexhatárral adja meg, a feladatra max. **6,5 pont**ot kaphat.)

	A	B	C	D	E	F	G	H	I
1	Teljes érték (EFt)	253			1.félévi támogatók			2.félévi támogatók	
2	Az egyes iskolák része(Ft)				Balogh	22		Daróci	14
3	Focisuli	Képző	Alap		Cecey	15		Kiss	35
4	101200	63250	88550		Fodor	25		Nagy	13
5					Nagy	15		Papp	24
6					Tóth	32		Szabó	43
7					Zalai	15		Összes(Eft)	129
8					Összes(Eft)	124			

2. Készítsen ***FelevFt*** nevű paraméteres ***Sub***-ot egy adott félév adományozói nevének és adományainak kiiratására, valamint az adományok összegzésére, és ezen összegek kiiratására.

A ***FelevFT*** formális paraméterei legyenek az ***Integer*** típusú ***melyik***, ***n*** és ***osz***, a ***String*** típusú ***Kitol*** tömb, valamint az ***Integer*** típusú ***Mennyi*** tömb, melyek rendre az adott félév számjelének (1 vagy 2), a támogatók (teljes) számának, a kiiratás kezdő oszlop-számának, valamint a neveket ill. adományokat tartalmazó tömböknek felelnek meg.

Hívja meg a ***FelevFt Sub***-ot a ***Tamogat*** program kétszer, úgy, hogy az egyes hívások eredményül a fenti táblázat ***E-F*** ill. ***H-I***. oszlopait adják. **(7,5 pont)**

6.4.5.4 2. feladatsor, B csoport

Név:	tk:	Z1 (2012. nov.29.)	B

A jobbra látható ***Pogacsa.txt*** fájl első sorában a két gyártott pogácsa nevét, majd a ***Figyelt anyag (kg/kg)*** szöveg után a figyelt alapanyagok számát, neveit és az egyes pogácsa fajták 1 kg-jához szükséges mennyiségeit adja meg. Ezután a ***Rendelések(kg)*** szöveget az egyes pogácsa fajták napi rendelési értékei követik. Az alább megadott feladatokat úgy oldja meg, hogy a program olyan (a ***Pogacsa.txt*** fájllal azonos szerkezetű) input fájl esetén is helyes eredményt adjon, ahol a figyelt anyagok száma és a rendelések napjainak a száma más, de a figyelt anyagok száma <= 5, és a rendelések napjainak a száma <=30.

```
Sajtos, Túrós
Figyelt anyag (kg/kg), 3
Liszt,  0.7, 0.6
Vaj,    0.1, 0.15
Tejföl, 0.1, 0.15
Rendelések(kg)
10, 0
 0, 10
10, 10
15, 20
12, 14
10, 20
```

1. Készítsen ***Pogi*** nevű VBA programot a fájl adatainak beolvasására, és az adatokból az egyes napokra vonatkozó alapanyag fogyasztás kiszámítására és kiiratására az jobbra látható táblázat ***A, B, C, D*** oszlopa szerinti formában. A pogácsa fajták nevét a ***PogiNev$(2)*** – ként, a rendelések kg értékeit a ***rend#(30,2)*** – ként deklarált tömbökbe olvastassa, ez utóbbinál a második index 1 értéke az első (a Sajtos), 2 értéke a második (a Túrós) pogácsa rendelési kg-jára vonatkozik. **(8 pont)**

	A	B	C	D	E	F	G
1	Napi fogyás(kg)					Összes kg 6 nap alatt	
2	nap	Liszt	Vaj	Tejföl		Sajtos pogácsa:	57
3	1	7	1	1		Túrós pogácsa:	74
4	2	6	1,5	1,5			
5	3	13	2,5	2,5			
6	4	22,5	4,5	4,5			
7	5	16,8	3,3	3,3			
8	6	19	4	4			

2. Készítsen ***OsszFogy*** nevű, ***Integer*** típusú ***Function***-t az adatfájlban megadott napokon összesen megrendelt, adott fajtájú pogácsa mennyiségének a kiszámítására. A ***Function*** neve az adott fajtájú pogácsa kg-ban számított mennyiségét vegye fel.

A ***Function*** formális paraméterei legyenek az ***Integer*** típusú ***a*** és ***n***, valamint a ***Double*** típusú ***x*** tömb, melyek rendre a pogácsaneveket tartalmazó tömb index-értékének, az összes rendelési nap számának, valamint a rendelési kg-okat tartalmazó tömbnek felelnek meg.

Hívja meg az ***OsszFogy Function***-t a ***Pogi*** program kétszer úgy, hogy az egyes hívások eredményül a fenti táblázat ***G2 ill. G3*** cellákat adják. A ***Pogi*** program hajtsa végre az ***F*** oszlopban látható 3 kiírást is. **(7 pont)**

6.4.5.5 3. feladatsor, A csoport

Név: **Tankör:** **Diger-3, A csoport – 2012 ősz**

1. Írja meg az „A1” programot, amely
a) először az Utazas.txt file-ban lévő ismeretlen számú adatpárokat olvassa be a „Varos” (sztring típusú) és „Ut” (integer típusú) nevű dinamikus tömbökbe a városok nevét, illetve a menettérti autóút hosszúságát (használja az EOF kifejezést!), miközben a „db” változóba rögzíti a beolvasott adatpárok számát,
b) majd az Inputbox utasítás segítségével bekéri a „Uticel” változóba (sztring) a keresendő város nevét.
c) Ezután a program keresse ki a „Varos” dinamikus tömb elemei közül azokat, amelyekre igaz, hogy Varos(i) = Uticel („i” ciklusváltozó). A program a „Celszam” (integer) változóban tárolja a találatok számát, illetve Uticelossz (integer) változóban rögzítse az adott városba történt utazások számának megfelelő összes megtett km értékét.
d) Végezetül az alábbi ábrán látható módon írja ki az eredményt. (11.5 pont)

	A	B
1	Utazások száma:	18
2	Célpont:	Pécs
3	Célpontutazások (szam):	4
4	Célpontutazások (km):	1584

```
Fájl  Szerkesztés  Beállítá
Szolnok, 194
Győr, 247
Pécs, 394
Nyíregyháza, 490
Veszprém, 221
Veszprém, 219
Kaposvár, 378
Pécs, 397
Eger, 255
Debrecen, 452
Győr, 245
Tatabánya, 112
Pécs, 396
Veszprém, 220
Szeged, 341
Pécs, 397
Szeged, 343
Eger, 257
```

2. a) Írja meg a „A2” program azon részét, mely az „y” (double) változójának értékadásakor meghívja az „Faktorialis” függvényt, melyhez egyetlen paraméter, az „x” (integer) változó tartozik („x” és „y” deklarálása nem szükséges). Az „y” értékét MsgBox paranccsal jelenítse meg.
b) Írja meg a double típusú „Faktorialis” függvényt, mely a „A2” program „x” paraméterét veszi át a függvény „xbe” integer típusú változójába és kiszámítja a

$$\prod_{i=xbe}^{1} i = xbe \cdot (xbe-1) \cdot (xbe-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$ értékét („i” ciklusváltozó),

amit majd átad a főprogram „y” paraméterének. (3.5 pont)

6.4.5.6 3. feladatsor, B csoport

Név: **Tankör:** **Diger-3, B csoport – 2012 ősz**

1. Írja meg az „B1” programot, amely
a) először az Vasarlas.txt file-ban lévő ismeretlen számú adatpárokat olvassa be a „Aru” (sztring típusú) és „Kiadas” (integer típusú) nevű dinamikus tömbökbe a vásárolt árúk nevét, illetve a fizetett összeget (használja az EOF kifejezést!), miközben a „db” változóba rögzíti a beolvasott adatpárok számát,
b) majd az Inputbox utasítás segítségével bekéri a „Tetel” változóba (sztring) a keresendő tétel nevét.
c) Ezután a program keresse ki a „Aru” dinamikus tömb elemei közül azokat, amelyekre igaz, hogy Aru(i) = Tetel („i” ciklusváltozó). A program a „Tetelszam” (integer) változóban tárolja a találatok számát, illetve „TetelFt” (integer) változóban rögzítse az adott tétel vásárlására költött összeget.
d) Végezetül az alábbi ábrán látható módon írja ki az eredményt. (11.5 pont)

	A	B
1	Kiadások száma:	20
2	Vásárlás (tétel):	élelmiszer
3	Vásárlás (szam):	3
4	Vásárlás (Ft):	12185

```
Fájl  Szerkesztés  Beállítások
BKV bérlet, 9800
újság, 175
élelmiszer, 2870
telefonszámla, 3200
tisztítószer, 1570
DVD film, 990
írószer, 1255
menza, 890
élelmiszer, 4150
könyv, 2480
fogkrém, 285
újság, 310
menza, 1120
DVD film, 1980
Főgáz számla, 2960
élelmiszer, 5165
Elmü számla, 5690
könyv, 1565
menza, 920
mozijegy, 1290
```

2. Írja meg a „B2” program azon részét, mely az „y” (double) változójának értékadásakor meghívja az „Egyperiksz” függvényt, melyhez egyetlen paraméter, az „x” (integer) változó tartozik („x” és „y” deklarálása nem szükséges). Az „y” értékét MsgBox paranccsal jelenítse meg.
b) Írja meg a double típusú „Egyperiksz” függvényt, mely a „B2” program „x” paraméterét veszi át a függvény „xbe” integer típusú változójába és kiszámítja a

$$\sum_{i=1}^{xbe} \frac{1}{i} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{xbe-2} + \frac{1}{xbe-1} + \frac{1}{xbe}$$ értékét („i” ciklusváltozó),

amit majd átad a főprogram „y” paraméterének. (3.5 pont)

6.4.5.7 4. feladatsor, A csoport

Név: **Tankör:** **Diger-4, A csoport – 2012 ősz**

1. Egészítse ki a baloldali programrészletet a jobb oldali oszlopban lévő 1-16. számú részletekkel. (8 pont)

```
Sub Feladat1___

___ i%, j___, x___

___:
x = ___("x=?")
If x ___ -3.5 ___x > 3.5 Then ___

For i = 1 To 4
   Cells(1, i + 1) = i
   Cells(i + 1, 1) = i
   For j = 4 To 1 ___ -1
      If i > j Then
         Cells(i + 1, j + 1) = i * x + j
         ___j > i ___
         Cells(i + 1, j + 1) = i * x - j
         ___
         Cells(i + 1, j + 1) = i * x
      ___
   ___
___

End Sub
```

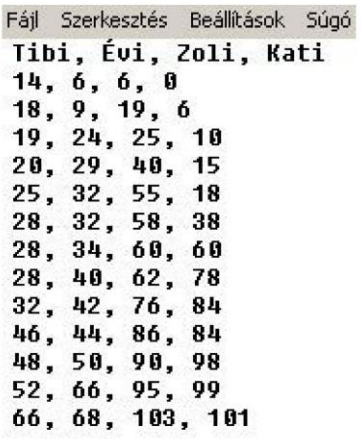
1) ElseIf
2) ujra
3) Step
4) Next j
5) ()
6) InputBox
7) %
8) Next i
9) GoTo ujra
10) Dim
11) Then
12) !
13) Else
14) Or
15) End If
16) <

2. Írja meg a Feladat2 VBA programot, amely:
a) először az ábrán látható munkalapról beolvassa a „Hitel”, „Kamat” és „Reszlet” változókba (mind double) a felveendő hitel nagyságát, havi kamatát és a havi törlesztő részletet. A „Hitel” változó értékét adja át a „Maradek” változónak (double),
b) leellenőrzi, hogy a havi törlesztés meghaladja-e hitel havi kamatterhét. Amennyiben nem, akkor MsgBox üzenettel jelezze, hogy a hitel nem visszafizethető ilyen alacsony havi törlesztéssel.
c) ha a hitel visszafizethető, a „Maradek” változó értékét növelje meg a havi kamatteherrel és vonja le a törlesztőrészletet. Majd a „Torlesztes” változó (double) értékét pedig növelje meg a törlesztő összeg értékével. Ezt a ciklust addig csinálja, amíg a „Reszlet” nagyobb nem lesz a „Maradek” változónak kamatteherrel megnövelt értékénél. A program közben számolja a törlesztési hónapok számát („Honap” változó, integer). Ha a havi részlet nagyobb a „Maradek” változónak kamatteherrel megnövelt értékénél, a „Torlesztes” változó értékéhez csak ezt az összeget adja hozzá.
d) végezetül az eredményeket a munkalapon látható módon jelenítse meg (a „Különbség” a törlesztett és a felvett összeg különbségét jelenti). (7 pont)

	A	B
1	Hitel (Ft) =	1000000
2	Havi kamat (%)=	1
3	Részlet (Ft) =	20000
4		
5	Időtartam (hónap) =	69
6	Törlesztés (Ft) =	1393237
7	Különbség (Ft) =	393237

Változók és jelentésük:
Hitel: a teljes felvett pénz mennyiség (double),
Kamat: a havi kamat (double),
Reszlet: a havi törlesztés összege, vagyis a havi befizetés (double),
Torlesztes: a visszafizetett pénz összege (double),
Maradek: a teljes, még vissza nem fizetett pénz összege (double),
Honap: hónapok száma, amelyekben befizetés történt (integer).

3. Írja meg a Feladat3 VBA programot, amely:



a) a „Kartya.txt” file-ból (jobboldali ábra) a „Nevek” tömbbe (index: 4, string típusú) beolvassa a „Fekete macska” kártyajátékban résztvevő személyek nevét, majd a „Pontok” 4*20-as tömbbe (integer) a partikban szerzett pontok összegét, miközben az „osztas” változóban (integer) számolja meg a leosztások (partik) számát,

b) a neveket, a pontokat, valamint a játékok számát az alsó ábrán látható módon kiírja egy Excel munkalapra,

c) a felhasználótól a „J1” és „J2” változókba (integer) bekéri két játékos számát (pl.: Tibi: 1, Évi: 2),

d) meghívja „Kulonbseg” szubrutint, melynek a főprogram átadja a „Pontok”, „osztas”, „J1” és „J2” változókat a szubrutin „Pontokbe” (tömb), „osztasbe”, „J1be” és „J2be” változóiba (mind integer) és „Min” és „Max” (mindkettő integer) változókat, melyekbe a szubrutin a kiszámolt „Minki” és „Maxki” (mindkettő integer) változókat adja vissza. A „Kulonbseg” szubrutin a két játékos pontjai közötti legnagyobb eltéréseket határozza meg, ezeket rögzíti a „Minki” és „Maxki” változókban.

e) „J1” és „J2” játékosok nevét, valamint a köztük lévő legnagyobb különbségeket az ábrán látható módon írja ki az Excel munkalapra (a -14 a 12. leosztás, a 9 a 2. leosztás után alakult ki). A cellákba történő kiíráskor a sorindexet az „osztas” változó függvényében adja meg!

A programozás során használhat még ciklusváltozókat (i, j, k, mind integer), illetve a „Kulonbseg” szubrutinban lokális változókat is, ha szükségesnek tartja. (15 pont)

	A	B	C	D	E	I
1		Tibi	Évi	Zoli	Kati	
2	1	14	6	6	0	
3	2	18	9	19	6	
4	3	19	24	25	10	
5	4	20	29	40	15	
6	5	25	32	55	18	
7	6	28	32	58	38	
8	7	28	34	60	60	
9	8	28	40	62	78	
10	9	32	42	76	84	
11	10	46	44	86	84	
12	11	48	50	90	98	
13	12	52	66	95	99	
14	13	66	68	103	101	
15						
16	1. játékos (Jat1): Tibi					
17	2. játékos (Jat2): Évi					
18	Jat1-Jat2 legnagyobb pontkülönbségek:					
19	-14	9				

7 Függelék

A függelék olyan Visual Basic for Application ismereteket tartalmaz, melyek nem szükségesek a tárgy teljesítéséhez, de az egyetemi évek alatt kiadott feladatok programozással történő megoldása során hasznosak lehetnek a hallgatók számára.

7.1 Rekordok és használatuk

Irodalom: Peter G. Aitken: Programozás VISUAL BASIC 6 nyelven, „'Kék Könyv" / 372 old. -

7.1.1 Fájlok írása és olvasása

Ha adatainkat el akarjuk menteni, hogy a gép kikapcsolása után is megmaradjanak, akkor a lemezen tároljuk azokat. Ha olyan adatokat akarunk felhasználni, amit egy másik program hozott létre, akkor általában fájlból kell azokat beolvasnunk. A Basic nyelv számtalan utasítást és függvényt tartalmaz, melyek rendkívül rugalmassá teszik a fájlok használatát. Mivel ezek az utasítások és függvények a Basic nyelv részét képezik, ezért nincsenek közvetlen kapcsolatban a Visual Basic felhasználói felületével, illetve annak objektumaival és eseményeivel.

A Visual Basicben egy fájl használata három lépésből tevődik össze:

- A fájl megnyitása.
- Adat olvasása a fájlból, adat írása fájlba.
- A fájl bezárása.

Ez így elég egyszerűen hangzik, de e három lépés során számos nehézség merülhet fel. Először vessünk egy pillantást a Visual Basicben használható három különböző fájlelérési módra.

7.1.1.1 A fájlelérés típusai

Minden fájlban lévő adat, akárhonnan is származzon, akármi is legyen a tartalma, bájtsorozatként tárolódik. Az adat típusától, illetve a program igényeitől függően három lehetséges fájlelérési mód közül választhatunk. A programunk által használt fájlelérési mód nagymértékben befolyásolja, hogy milyen technikákkal olvashatunk és írhatunk adatot, és egyáltalán azt, hogy mit kezelünk adatként

7.1.1.2 Szekvenciális elérésű fájlok

Egy szekvenciális elérésű fájl változó hosszúságú rekordok sorozatát tárolja. Itt a rekord szó egyszerűen egy adategységet jelent. Például, ha egy **Integer** típusú változó értékét tároljuk a fájlban, akkor az egy rekord. Hasonlóan egy **String** típusú változó értéke egy másik rekord lesz. A rekordok változó hosszúságúak, mivel minden rekord méretét a tartalma határozza meg, nem pedig a fájl általános típusa. Más szóval a szekvenciális fájlok változó hosszúságú rekordokat tartalmazhatnak.
A szekvenciális fájlokban lévő adatot szövegnek tekintjük. Ez annyit jelent, hogy a fájl minden bájt ja egy karakter. A szekvenciális szó azt jelzi, hogy az ilyen típusú fájlt mindig az elejéről kell olvasni, ahhoz, hogy a fájl közepéből olvashassunk, először ki kell olvasni minden azt megelőző elemet. Nincs egyszerű módja annak, hogy egy meghatározott sorszámú adatra ugorjunk. A szekvenciális fájlokban a numerikus (számszerű) adatok a hozzájuk tartozó karaktersoroztokkal kerülnek tárolásra, a sztring adatok pedig macskakörmök között, Például a 123 nem bináris adatként, egy bájton, hanem a „123" szövegként kerül tárolásra.

A szekvenciális elérésű fájlokban minden sort egy rekordnak tekintünk, melyet egy. kocsi-vissza" + "sor-emelés" karakterpár zár (CR+LF). Egy rekord tartalmazhat több, de akár nulla karaktert is, és a rekordok hossza nem függ a többi rekordtól. Ez a meg- közelítés akkor lehet hasznos, ha eltérő hosszúságú adatokkal kell dolgozni, és csak szekvenciális feldolgozást akarunk megvalósítani.

7.1.1.3 Véletlen elérésű fájlok

Egy véletlen elérésű (random access) fájl állandó hosszúságú rekordok sorozataként tárolja az adatokat (a rekordok mérete állandó). A véletlen elérésű fájlban a rekordok sorszámokkal rendelkeznek. A sorszámozás egytől kezdődik. E fájlokban egy megadott sorszámú rekordot azonnal elérhetünk, anélkül, hogy az előtte lévő adatokon végig kellene, fussunk.

Például egy 2000 rekordból álló fájlból kiolvashatjuk az 1199-dik rekordot anélkül, hogy előzőleg kiolvasnánk az összes többit 1-tőI1198-ig. A véletlen elérésű fájlokban szöveget és számokat egyaránt tárolhatunk. A szöveg karakteresen, a számok pedig speciális bináris formában tárolódnak.

A véletlen elérésű fájlokban minden rekord azonos számú bájtból, illetve karakterből áll. Minden rekord tartalmaz egy vagy több mezőt, melyek közül mindegyiknek rögzített a hossza. Ebből következik, hogy egy rekord mérete megegyezik a mezők hosszainak összegével. A fájl rekordjainak, illetve a rekordok mezőinek hosszát a programban határozzuk meg, a fájllétrehozásakor. Eltérően a szekvenciális fájloktól, a véletlen elérésű fájlok nem használnak szeparátorokat (olyan jeleket, melyek elválasztják a rekordokat egymástól, mint például a szekvenciális fájlnál a CR+LF karakterpár) az egyes mezők és rekordok kezdetének és végének jelölésére. Az állandó hosszúság teszi lehetővé, hogy a fájlban egy tetszőleges sorszámú rekordra ugorhassunk.

Például ha a rekordméret 100 bájt, akkor tudni lehet, hogy az 55. rekord az 5401. bájtnál kezdődik.

7.1.1.4 Bináris fájlok

A bináris fájlok esetében a fájlok szerkezetére semmi sincs megadva. Semmiféle rekordméret, vagy szeparátor nincs kikötve, a fájlt egyszerűen csak bájtok sorozatának tekintjük. A bináris hozzáférés lehetővé teszi, hogy a fájl egyes bájtjait a fájl típusától függetlenül manipuláljuk. Eltérően a véletlen elérésű és a szekvenciális fájloktól, a bináris fájlok tartalma nem csupán ASCII szöveg lehet. A bináris hozzáféréssel bár milyen fájlt olvashatunk, írhatunk, illetve módosíthatunk, viszont nekünk kell kezelni, hogy a fájlban mit és hogyan tárolunk.

7.1.2 Szekvenciális (SOROS hozzáférésű) fájlok használata

Amint azt a korábbiakban kifejtettük, a szekvenciális hozzáférésű fájlok, rekordok sorozatát táolják, ahol minden rekord egy szövegsor, melyet egy CR+LF (kocsi vissza + soremelés) karakterpár zár le. Itt emlékeztetünk arra, hogy a szekvenciális fájlok egyszerre csak egy műveletre nyithatók meg: olvasásra (INPUT mód), vagy írására. Ez utóbbi esetben megkülönböztetendő az új fájl írása (OUTPUT mód), illetve létező fájl végére történő írás (APPEND mód), Ahhoz, hogy szekvenciális fájloknál az egyik műveletről a másikra váltsunk - például írásról olvasásra - először be kell zárni a fájlt, majd megnyitni a másik módban.

7.1.2.1 Szekvenciális hozzáférésű fájlok írása és olvasása

Ha a tárolandó információ azonos szerkezetű diszkrét (a többi adattói jól elkülöníthető) egységekből áll, akkor használhatunk mezőket a szekvenciális fájlban. Példa lehet erre egy könyvek adatait nyilvántartó fájl. Minden könyvhöz tárolni akarjuk az íróját, a címét, és egy indexet, amely a könyv helyét határozza meg a polcon. Ehhez készíthetünk egy szekvenciális fájlt, melynek szerkezete a következő:
A fájlban minden rekord egy-egy könyv adatait tartalmazza.
Minden rekord három mezőből áll, melyek közül az első a szerző, a második a cím, a harmadik az index.
Legyen az első bejegyzés a "A nagy mesekönyv" című könyv, melyet Benedek Elek írt, az indexszáma pedig 12. Ehhez a bejegyzéshez tartozó rekord a fájlban a következőképpen fog kinézni:

„Benedek Elek", "A nagy mesekönyv", 12

Ha a második bejegyzés Dosztojevszkijtől a Bűn és bűnhődés, melynek indexszáma 123, akkor az a következőképpen tárolódik:

"Dosztojevszkij","Bűn és Bűnhődés", 123

Minden rekord külön sorban lesz. Ezekből a példákból világosan látszik, hogy szekvenciális fájlokban vesszők határolják (vesszőt használ szeparátorként) az egyes mezőket, a szöveges adatok pedig macskakörmök között vannak.
Ha egy rekordot akarunk kiírni egy szekvenciális fájlba (szeparátorokkal együtt), akkor a Write # utasítást kell használnunk:

Write # filenum, [lista]

- filenum: az a szám, melyet a fájl- OUTPUT vagy APPEND módban való megnyitásakor a fájlhoz rendeltünk.
- lista: egy vagy több Basic kifejezés, melyet a fájlba akarunk írni. Ha a lista több mint egy elemet tartalmaz, akkor azokat vesszővel kell elválasztanunk. Ha nem adunk meg listát a Write # utasításnak, akkor üres rekord (tehát üres sor) kerül a fájlba.

Íme, egy egyszerű példa:

```
Write #filenum, "Benedek Elek", "A Nagy Mesekönyv ", 12
```

Egy programban általában változókkal paraméterezzük a Write # utasítást:

```
book.name="Benedek Elek"
book.title="A Nagy Mesekönyv"
book.index=12
Write #filenum, book.name, boook.title, book.index
```

Minden Write # utasítás egy rekordot ír a fájlba úgy, hogy automatikusan beszúrja a szükséges vesszőket és macskakörmöket. Az ilyen fájlba kerülő adatokban nem szerepelhet a macskaköröm.

Szekvenciális fájlból a mezőkre osztott adatot az Input # utasítással lehet beolvasni.

Az Input # beolvas egy vagy akár több mezőt a fájlból, majd azokat programváltozókba írja:

```
Input # filenum, lista
```

Ahol az egyes paraméterek jelentése a következő:

filenum*:* a fájl- INPUT módban történő - megnyitásakor a fájlhoz rendelt szám.

lista: egy vagy akár több programváltozóból álló lista. Ezekbe kerülnek a fájlból kiolvasott adatok. Ha a lista egynél több változónevet tartalmaz, akkor azokat vesszővel kell elválasztani.

Az Input # utasítás annyi mezőt olvas be a fájlból ahány változó meg van adva a paraméterlistában, az egyes mezőket pedig sorrendben a megfelelő változókhoz rendeli. Ne felejtsük el, hogy a szekvenciális fájlokat csak szekvenciálisan lehet olvasni, azaz csak az első rekord első mezőjéről lehet indulni, és onnan lehet haladni előre.

Íme egy egyszerű példa:

```
        filenum = FreeFile
Open "Konyv,txt" For Output As # filenum
        Write # filenum, "Benedek Elek", "A Nagy Mesekönyv", 12
        Write # filenum, "Dostoyevski", "Bűn és Bűnhődés", 123
Close # filenum
```

Majd a program egy másik részében így olvashatjuk vissza:

```
Dim A As String, B As String, C As String, O As String
Dim x As Integer, y As Integer
        filenum = FreeFile
Open "Konyv.txt" For INPUT As # filenum
        Input # filenum, A, B, x
        Input # filenum, C, O, Y
Close # filenum
```

A fenti kód lefutása után az egyes változók értékei a következők lesznek:

A=" *Benedek Elek* ", B=" *A Nagy Mesekönyv* ", x=12,

C="Dostoyevski", D=" *Bűn és Bűnhődés*", y=123.

Ha a fenti kódban kicseréljük a két Input # utasítást a következő, egy Input # utasításra:

```
Input # filenum, A, B, x, C, O, Y
```

vagy akár az alábbi három utasításra:

```
Input # filenum, A, B
Input # filenum, x, C
Input #filenum, O, y
```

akkor is ugyanezt az eredményt érjük el. Az, hogy az Input # utasítás miként bontja a fájlt mezőkre, az a paraméterlistában szereplő változók típusától függ. Ha az utasítás éppen String típusú változóba olvas, akkor a mező végét a következő jelek valamelyike mutatja:

- Idézőjel, ha mező Idézőjellel kezdődött.
- Vessző, ha a mező nem idézőjellel kezdődött.
- Kocsi vissza + soremelés (CR+LF) karakterpár

Ha numerikus változóba olvas, akkor a mező végét az alábbi karakter sorozatok valamelyike jelzi:

- Egy darab vessző.
- Egy vagy több szóköz.
- Kocsi vissza + soremelés (CR+LF) karakterpár

Ha szekvenciális fájlban tárolunk adatokat, akkor a programban ügyelni kell arra, hogy a Write # és az Input # utasítások az egyes rekordok mezőinek típusát, sorrendjét, és számát tekintve összhangban legyenek. Más szóval az egyes adatelemeket ugyanolyan sorrendben kell kiolvasni a fájlból, ahogy beleírtuk azokat.
Ha ez az összhang felborul, akkor kétféle probléma is felmerülhet:

- Előfordulhat, hogy a program az Input # utasítással a fájlból egy mezőt egy attól különböző típusú változóba akar beolvasni (például sztringet olvas egy numerikus változóba). Ez nem feltétlenül okoz hibát, de valószínűleg váratlan eredményeket produkál. Ha numerikus mezőt olvasunk be sztring változóba, akkor a változó az adott szám szöveges reprezentációját kapja értékül. Ha sztring mezőt olvasunk be numerikus változóba, akkor az eredmény a sztring tartalmától függ. Ha a sztring nem numerikus karakterrel kezdődik, akkor a változó értéke nulla lesz, egyébként a változó a sztring elején található szám értékét veszi fel.
- Ha a mezők száma nem azonos, akkor előfordulhat, hogy a program "eltéved" a fájlban, azaz nem tudja, hogy éppen melyik rekordnál tart. Emlékezzünk rá, hogy az Input # mezőket, és nem rekordokat számlál az olvasás során. Tehát a végrehajtott Input # utasítás a következő mezőnél fogja elkezdeni az olvasást, ami viszont nem biztos, hogy a következő rekord elejével azonos.

A szekvenciális elérésű fájlok egyik felhasználási területe a változó hosszúságú sztringekből álló tömbök tárolása. A most következő kódrészlet azt mutatja be, hogy hogyan kell az efféle tömböket fájlba írni:

```
Dim Nevek(100) As String, Szam As Integer, filenum As Integer
```

(a Nevek(100) tömböt először adatokkal kell feltölteni, különböző hosszúságú nevekkel).

```
        filenum = FreeFile
Open "NOTES.TXT" For OUTPUT As #filenum
        For Szam = 0 to 100
                WRITE #filenum, Nevek(Szam)
        Next Szam
Close # filenum
```

Az alábbi pedig visszaolvassa az adatokat a fájlból, és belepakolja egy sztringekből álló tömb be:

```
Open "NOTES. TXT" For INPUT As #filenum
        For Szam = 0 To 100
                Input #filenum, Nevek (Szam)
        Next Szam
Close #filenum
```

Szekvenciális fájlokban számokból álló tömböt is tárolhatunk, de amint a későbbiekben látni fogjuk, erre a célra egy másik elérési mód előnyösebb.

7.1.2.2 A fájl végének vizsgálata

Ha a program szekvenciális fájl ból olvas adatot, akkor az EOF függvény segítségével le lehet kérdezni, hogy a program elérte-e a fájl végét:

```
EOF (filenum)
```

Az EOF igazat (True értéket) ad vissza, ha a fájlból már kiolvastuk az utolsó rekordot, hamisat (False), ha még nem. Ha egy szekvenciális fájlból az utolsó rekord kiolvasása után még olvasni próbálunk, akkor hiba keletkezik. A most következő kód mutatja meg, hogy miként lehet egy ciklussal, az EOF függvényt használva beolvasni egy teljes szekvenciális fájlt soronként, majd egy tömbbe rakni:

```
Dim info(1000) As String
Dim Szam As Integer

Open "Sajat.txt" For Input As #filenum
        Szam = 0
        Do While Not EOF(filenum)
                Input #filenum, info (Szam)
                Szam=Szam+1
        Loop
Close # filenum
```

7.1.3 Véletlen elérésű fájlok használata

A véletlen elérésű fájl annyiban hasonlít szekvenciális társához, hogy mindkettő rekordokból áll. A véletlen elérésű fájl rekordjainak viszont egyforma hosszúaknak kell lenniük, hiszen a program az egyes rekordokat fájlbeli pozíciójuk alapján azonosítja. A rekordokat nem választja el speciális karakter. Például ha a fájl minden rekord-ja 100 bájt hosszú, akkor az első rekord a fájl első bájtjától a századikig található (a fájlokban a bájtok sorszámozása 1-gyel kezdődik. A negyedik rekord pedig a 301. pozíción kezdődik, és a 400-ig tart. A véletlen elérésű fájl rekordjai eggyel kezdve, egyesével növekvően vannak számozva. Igazából nem is számozva vannak, hanem a fájlban elfoglalt helyük határozza meg, hogy ők hányadikak. Legfeljebb 2 147 483 647 rekord lehet egy fájlban, ennél többet ilyen módon nem tudunk kezelni. Minden rekordban több mező lehet. Egy véletlen elérésű fájlban minden rekordnak azonos a szerkezete. Más szóval ugyanannyi a mezők száma, és ugyanaz az egyes mezők mérete. Mielőtt létrehozunk, vagy használunk egy véletlen elérésű fájlt, először meg kell adni annak rekordszerkezetét.

7.1.3.1 A véletlen elérésű fájlok rekordszerkezetének megadása

A véletlen elérésű fájlok rekordszerkezetének megadása ugyanúgy történik, mint a felhasználói adattípusok definiálása, azaz a Type ..• End Type ***kifejezéssel.***

*(Ezt az **5.7** Felhasználói Adattípusok fejezetben tárgyaltuk részletesen. 21. old.)*

Első lépésben készítünk egy olyan adattípust, melynek szerkezete megegyezik a létrehozandó véletlen elérésű fájl rekord-szerkezetével. A könyvnyilvántartó program esetén ez például lehet a következő:

```
Type Konyv_reg
        Szerzo As String * 20
        Cim As String * 25
        Index As Integer
End Type

Dim Könyv_tar(100) As Konyv_reg
```

Most van egy felhasználói típusunk, ami képes lesz kezelni a könyvekre vonatkozó adatokat a programban, továbbá meghatározza az adatfájl rekordszerkezetét, Véletlen elérésű fájlok megnyitásakor az egyik paraméter a fájl rekordjainak hossza (amint azt a fejezet egy korábbi részében láttuk az Open utasítás tárgyalásakor). Ezt a számot könnyen megkaphatjuk, ha meghívjuk a Len függvényt, mely visszaadja a paraméterként kapott típus hosszát bájtokban. Tehát az Open utasítás a következőképpen néz ki:

```
    filenum = FreeFile
Open "Konyvtar.txt" For RANDOM As #filenum Len = Len (Konyv_tar(1) )
```

Vegyük észre, hogy a Len függvénynek a tömb egy elemét adtuk át, és nem az egész tömböt. Ez az utasítás a felhasználói adatszerkezet méretével, azonos rekordhosszal nyitja meg a véletlen elérésű fájlt.

7.1.3.2 Véletlen elérésű fájlok írása és olvasása

Véletlen elérésű fájlokba írni a **Put** utasítással lehet:

```
    Put #filenum, [rekordsorszám], felhasználóiTípus
```

Ahol az egyes paraméterek jelentése a következő:

filenum: a fájl megnyitásakor a fájlhoz rendelt szám

rekordsorszám: annak a rekordnak a sorszáma, ahová az adatot akarjuk beírni. Ha nem adunk meg rekordsorszámot, akkor a soron következő helyre kerül az adat (az a legutolsó Get vagy Put művelet helyét követő pozícióra).
Ha a fájl megnyitása óta még nem volt Get vagy Put művelet, akkor a művelet az első rekordra vonatkozik.

- felhasználóiTípus: az a változó, mely a kiírandó adatokat tartalmazza, és szerkezete megegyezik a fájl rekordtípusával.

Folytatva a nyilvántartó példát, ha a fájl első rekordjába akarunk írni, akkor a következőt tehetjük:

```
Konyv_tar(1).Szerzo = "Benedek Elek"
Konyv_tar(1).Cim = "A Nagy mesekönyv"
Konyv_tar(1).Index = 12
Put # filenum, 1, Konyv_tar(1).
```

Amikor először létrehozunk egy véletlen elérésű fájlt, akkor kezdhetjük az írást az első rekordnál, mindig növelve a sorszámot eggyel minden egyes sikeresen végrehatott Put utasítás után. A Put utasításból akár el is hagyhatjuk a rekordsorszám paramétert, hiszen a Visual Basic automatikusan nyilvántartja az aktuális pozíciót. Viszont ha létező fájlhoz írunk hozzá újabb rekordokat, akkor magunk kell, hogy kezeljük a rekordsorszám paramétert, ezáltal biztosítva, hogy az adat tényleg oda kerüljön, ahová akarjuk. Ha egy olyan sorszámú rekordba írunk, mely már tartalmaz adatot, akkor az **Új adat** felülírja a régit. Ha nem akarunk felülírni már meglévő adatokat, akkor az új rekordokat a fájl végére kell írnunk. Más szóval, ha egy **n** rekordból álló fájlhoz írunk hozzá adatsorokat, akkor az új rekordok sorszámai **n+l**-nél kell, hogy kezdődjenek.
A Lof függvény segítségével határozhatjuk meg a fájlban lévő rekordok számát. Ez a A Lof függvény a fájl méretét adja meg bájtban:

```
Lof(filenum)
```

Ahol a filenum (fájlsorszám) paraméter annak a megnyitott fájlnak a száma, melynek a hosszát meg akarjuk tudni. Miért jó, hogy tudjuk a fájl hosszát bájtban? Tudjuk, hogy a Len függvénnyel a fájl egy rekordjainak hosszát kérezhetjük le, ebből következik, hogy a fájl hossza osztva egy rekord hosszával megadja a fájlban lévő rekordok számát. Tehát a megoldás, a következő record írására:

```
Rec_szam = (Lof(filenum) \ Len(Konyv_tar(1))) + 1
Put #filenum, Rec_szam, Konyv_tar(Rec_szam).
```

A következő olvasandó vagy írandó rekord sorszámát a Seek függvény adja meg.
Hasonlóan a Loc függvény, mely az utoljára olvasott, illetve írt rekord sorszámát adja vissza.
A Seek a fájlmutató aktuális pozícióját adja meg (azaz a következő olvasandó rekord számát),
a Loc pedig a legutoljára kiolvasott illetve beírt rekord helyét.
Szintaxisuk a következő:

```
Seek (filenum)
Loc (filenum)
```

A filenum (fájlsorszám) már megszokott jelentéssel bír.
A Loc által visszaadott érték eggyel kisebb, mint a Seek esetén.
Ha éppen most nyitottuk meg a fájlt, a Loc értéke 0, míg a Seek az 1-et adja vissza.

Véletlen elérésű fájlból olvasni a Get utasítással lehet. Az adat a fájl egy adott rekordjából kerül kiolvasásra, és az előre definiált felhasználói adattípusú változóba kerül. A Get utasítás szintaxisa:

```
Get [#]filenum, [rekordsorszám], felhasználóiTípus
```

Ahol az egyes paraméterek jelentése a következő:

- filenum: a fájl megnyitásakor a fájlhoz rendelt szám
- rekordsorszám: annak a rekordnak a sorszáma, ahová az adatot akarjuk beírni. Ha nem adunk meg rekordsorszámot, akkor a soron következő helyre kerül az adat (az a legutolsó Get vagy Put művelet helyét követő pozícióra).
  Ha a fájl megnyitása óta még nem volt Get vagy Put művelet, akkor a művelet az első rekordra vonatkozik.
- felhasználóiTípus: az a változó, mely a kiírandó adatokat tartalmazza, és szerkezete megegyezik a fájl rekordtípusával.

Folytatva példánkat, az adatfájl első rekordját a következőképpen olvashatjuk ki:

```
Get [#]filenum, 1, Konyv_tar(1)
```

Ha minden rekordot ki akarunk olvasni egy véletlen elérésű fájlból, akkor írjunk egy ciklust, mely az első rekordról indul, és folyamatosan olvas egészen a fájl végéig, és közben a kiolvasott elemeket belerakja egy megfelelő típusú tömbbe.

7.1.3.3 A fájl végének figyelése

A fájl végének a figyelés itt is az EOF függvénnyel lehetséges, úgy, mint a **Szekvenciális fájlok** végének figyelése, a - **7.1.2.2** fejezetben már tárgyalva lett Hasonlóan a fentiekhez, ebben az esetben is megkap hatjuk az utolsó rekord sorszámát, ha elosztjuk a fájl hosszát egy rekord hosszával.

A Visual Basic minden megnyitott véletlen elérésű fájlhoz nyilvántart egy mutatót. Ez a mutató határozza meg, hogy hányadik rekordot olvassa a Get, vagy hanyadikat írja a Put, ha nem adjuk meg a rekordsorszám paramétert. Egy véletlen elérésű fájl megnyitásakor - legyen akár új, akár létező fájl - a rekordmutató az első rekordra mutat. Véletlen elérésű fájlok írásakor és olvasásakor a fájlmutató a következőképpen módosul:
A Get és Put eljárások rekordsorszám paraméter nélküli hívása esetén a fájlmutató értéke eggyel nő.
A Get és Put eljárások rekordsorszám paraméterrel való meghívása esetén a fájlmutató rekordsorszám+1 lesz.

7.1.4 Bináris fájlok használata

A bináris fájlokat formázatlan bájtsorozatnak tekinthetjük. A fájlban nincs rekordszerkezet, ha mégis kell legyen, azt kódból szimuláljuk. A bináris fájlok nagyon rugalmas adattárolási lehetőséget biztosítanak, de strukturálatlanságuk miatt a programozónak kell "fejben tartania", hogy mi van a fájlban. Azt is nyilván kell tartanunk, hogy éppen hol tartunk a fájlban.

7.1.4.1 Fájlpozíció

Minden megnyitott fájlnak van egy fájlmutatója. A fájlmutató egy numerikus érték, amely a fájlnak arra a pozíciójára mutat, ahol a következő írási vagy olvasási művelet fog történni. A bináris fájlokban minden pozícióhoz egy bájt tartozik. Azaz, ha egy fájl **n** bájtot tartalmaz, akkor a pozíciók **1**-től **n**-ig számozódnak. A fájlmutatót megváltoztatni, illetve lekérdezni a Seek utasítással, illetve a Seek és Loc függvényekkel lehet. Ezek az utasítások ugyanúgy használhatók itt is, mint a véletlen elérésű fájloknál.
(A véletlen elérésű fájlok esetében viszont a mutató rekordokra mutat, és nem bájtokra.)
A Visual Basic tudja, hogy egy adott fájl milyen módban lett megnyitva, és ennek megfelelően értelmezi a Seek és Loc függvényeket. Bináris fájlok esetén bájtokra, véletlen elérésű fájlok esetén rekordokra. A Seek és Loc szintaxisa a következő:

```
Seek (#) filenum, újPozíció
```

A Seek utasítás a újPozíció pozícióra állítja a rekordmutatót
Ahol a filenum (fájlsorszám) a megnyitáskor a fájlhoz rendelt számot jelöli.

```
[poz =] Seek(filenum)
```

A Seek függvény pedig visszaadja a fájlmutató aktuális *pozícióját (azaz a következő írás vagy olvasás helyét.*

```
[poz =] Loc(filenum)
```

A Loc függvény az utoljára beolvasott vagy kiírt bájt pozícióját adja meg.
Ha nem írjuk saját kezűleg a mutatót, akkor a Loc mindig eggyel kisebb, mint a Seek.

7.1.4.2 Bináris fájlok olvasása - írása

Bináris fájlokból olvasni a Get utasítással lehet, írni beléjük pedig a Put utasítással. Ezek megegyeznek a véletlen elérésű fájloknál használt utasításokkal, ám másként értelmeződnek attól függően, hogy az írandó, illetve az olvasandó fájl milyen módban lett megnyitva:

```
Get [#] filenum, [poz], változó
Put [#] filenukm, [poz], változó
```

Ahol az egyes paraméterek jelentése a következő:

- filenum: a fájlhoz megnyitáskor hozzárendelt szám.
- poz: az olvasás, illetve írás helyét határozza meg. Ha nem adjuk meg a poz paramétert, akkor a fájlmutató által meghatározott helyen hajtódik végre a művelet.

- változó: egy akármilyen típusú Basic változó. A Get utasítás ebbe a változóba olvas a fájlból, a Put pedig ennek a változónak a tartalmát írja ki a fájlba. Az egyes utasítások automatikusan annyi bájtot olvasnak, illetve írnak amennyi bájtból a változó áll. Ha a változó egy változó hosszúságú sztring, akkor az átvitt bájtok száma megegyezik a ***változóban*** lévő karakterek számával.

Bináris fájlok megnyitásakor a fájlmutató az első pozícióra mutat. A Get **és** Put utasítások az átvitt bájtok számával növeli meg a fájlmutató értékét.

A bináris fájlok hosszát a Lof függvénnyel lehet meghatározni.

```
[hossz =] Lof(filenum)
```

A Lof által visszaadott érték megegyezik a fájl utolsó bájtjának pozíciójával. A Lof függvénnyel, és egy ciklussal gyorsan be lehet olvasni egy egész bináris fájlt. A következő részlet másolatot készít egy fájlról. A továbbiakban feltesszük, hogy az **Eredeti.txt** nevű fájl létezik, a **Masolat.txt** pedig nem.

Egy **1 karakterből** álló fix hosszúságú sztring segítségével a program mindig egy bájtot olvas, majd kiírja azt. Ez a kód természetesen működik, de azért megjegyzem, hogy a bájtonkénti olvasás „kissé" nehézkes, és kerülendő módszer.

```
Dim ch As String * 1, Szam As Integer, f1 As Integer, f2 As Integer

      f1 = freefile
Open "Eredeti.txt" For BINARY As #f1
            f2 = freefile
    Open "Masolat.txt" For BINARY As #f2
                    For Szam = 1 TO Lof(f1)
                    Get #f1, , ch
                   Put #f2, , ch
           Next Szam
Close #f1, #f2
```

Az EOF függvény segítségével lehet figyelni egy bináris fájl végét. Például a fenti kódban a ciklust a következőképpen is megírhattuk volna:

```
Do While Not Eof (f1)
   Get #f1, , ch
   Put #f2, , ch
Loop
```

A bináris fájlok rugalmasságuk miatt rendkívül jól használhatók a program adatainak tárolására. Ez különösen akkor igaz, amikor az adat formátuma nem illeszkedik sem a véletlen elérésű fájlok, sem a szekvenciális fájlok szerkezetéhez. Ne felejtsük el, hogy a bináris fájlokhoz nincs struktúra rendelve, ezért a programnak kell nyilvántartania az adattárolás formátumát. Például, ha a program egy ezer elemű tömböt *(dupla pontosságú változókat)* használ, akkor a szóban forgó tömböt a következőképpen **Bináris** fájlban tárolni:

```
Dim Data(999) As Double, Szam As Integer
```
(A Data tömböt előre fel kell tölteni, 1000 Double számmal)

```
Open "Ezres_tomb" For BINARY As # 1
    For Szam = 0 To 999
        Put # 1, , Data (Szam)
    Next Szam
Close #1
```

Adatainkhoz később a következőképpen férhetünk hozzá:

```
Open "Ezres_tomb" For BINARY As #1
    For Szam = 0 To 999
        Get #1, , Data2 (Szam)
    Next Szam
Close #1
```

Következzék néhány jó tanács bináris fájlok használatához:

Ha létező fájl végére akarunk új adatot írni, akkor állítsuk a mutatót a fájl végére a követező utasítással:

```
Seek (filenum), vagy  Lof (filenum) +1
```

Ha egy létező fájlban a fájl vége előtri pozícióra írunk adatot, akkor az **Új adat** felülírja a régit.

Ha a fájl vége utáni pozíciókra írunk adatot (például a Lof (ffilenum) +10 pozícióra) akkor a fájl meghosszabbodik a szükséges méretre **de a köztes területre szemét" kerül**, más szóval definiálatlan értékek.

Ha a fájl vége utáni bájtpozícióról próbálunk meg olvasni, akkor nem történik hiba, de a visszaadott adat-szemét", definiálatlan érték lesz. Ha nem figyelünk rá, elég érdekes adataink lesznek.

7.1.5 Melyik fájltípust használjuk?

Ha egy kis zavart érzünk a Visual Basic által nyújtott fájlelérési módokkal kapcsolatban, akkor megnyugodhatunk, nem vagyunk egyedül. Nem mindig teljesen világos, hogy melyik fájltípus a legmegfelelőbb egy adott alkalmazásban. A most következő tanácsokat segítségnek szánjuk a döntés meghozatalához:

- Ha az adat szövegsorokból áll, akkor - függetlenül attól, hogy azt egy Visual Basic alkalmazás hozta-e létre, vagy sem - használjunk szekvenciális elérést. A szekvenciális szövegfájlokat a hagyományos, és az FSO megközelítésben egyaránt elérhetjük.
- Ha olyan nem szöveges fájlokat akarunk használni, melyet más alkalmazások hoztak létre, és nem szigorú szerkezetűek, akkor használjuk a bináris elérést.
- Ha az adatnak pontosan megadható a rekordszerkezete, akkor használjunk véletlen elérést vagy szekvenciális fájlt. A programtói függően előfordulhat, hogy a szekvenciális a jobb.
- Numerikus adatok tárolásához használhatjuk a szekvenciális és a bináris módot is. A bináris mód annyiban jobb, hogy gyorsabb, és sokkal kisebb méretű fájlokat készít, de többet kell vele dolgoznunk.

- Változó hosszúságú sztringek tárolására használjuk a szekvenciális módot. Fix hosszúságú sztringek esetében használjunk vagy szekvenciális, vagy bináris fájlt. A bináris mód előnyösebb a sebesség, illetve a méret szempontjából, de ebben az esetben nem olyan jelentős a különbség, mint a numerikus tömbök esetében.
- A véletlen elérés esetében gyorsabban érhetőek el a fájl egyes rekordjai.
- Szöveges adat esetében viszont pazarló, mert az egyes sztringek szóközökkel egészülnek ki, hogy kitöltsék a rekordméretet.
- A szekvenciális mód esetében a fájl egyes rekordjainak elérése lassabb, viszont gazdaságosabb a helykihasználás, mert a sztringek nem egészülnek ki szóközökkel.

Ezek persze csak tanácsok. Ne tekintsük őket merev szabályoknak. Fontos viszont szem előtt tartani, hogy a fájlelérés módja nem változtatja meg a fájl fizikai szerkezetét. Csupán azt határozza meg, hogy a program miként olvassa, illetve írja a fájlt. Más szóval egy akármilyen fájl - függetlenül attól, hogy milyen módban nyitjuk meg mindenképp egy lemezen tárolt bájtsorozat. Például megtehetjük, hogy készítünk egy fájlt véletlen eléréssel, és mégis sokszor bináris módban nyitjuk meg. Azzal, hogy ugyanazt a fájlt több elérési móddal használjuk, gyakran drasztikusan lehet növelni a program hatékonyságát, vagy éppen csökkenteni az elkészítéséhez szükséges erőfeszítéseket. Amíg még csak ismerkedünk a fájlkezeléssel, addig mindenesetre egy fájlt csak egy móddal használjunk.

7.2 A For Each – Next ciklus és alkalmazása

7.3 A Solver alkalmazása a programírás során

7.4 A „User Form” használata