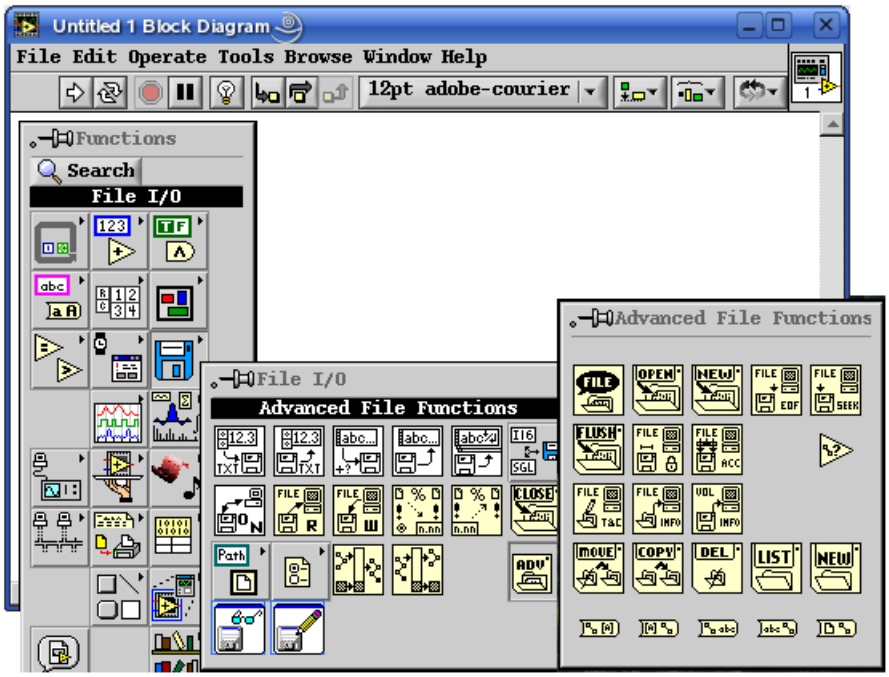
5. Példa: Olvasás szöveges fájlból (program3_1.vi)

A fájlműveletek során adatokat olvasunk ki vagy írunk be egy fájlba. Ez általában a következő három művelet elvégzését jelenti:

- Fájl megnyitása
- Adatok olvasása vagy írása
- Fájl bezárása

A LabVIEW rendszerben számos VI áll rendelkezésünkre a fájlműveletek megvalósításához (III.1. ábra). Ezek az eljárások három különböző szinten biztosítják ezen műveletek megvalósítását. A magas szintű eljárásokat tartalmazó VI-okra ('File I/O' paletta fehér ikonjai) az a jellemző, hogy csak egy speciális fájltípust tudnak írni illetve olvasni, de cserébe a fent említett mindhárom lépést (sőt még ha szükséges egy fájlmegnyitáshoz szükséges dialógusablak megnyitását is) elvégzik.

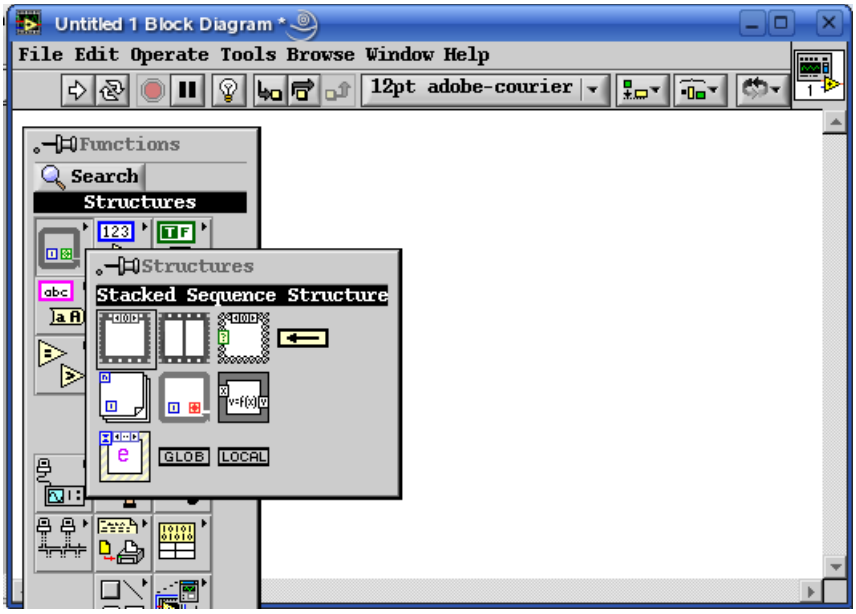


III.1. ábra: Fájlműveletek LabVIEW-ban

Középszintű fájlkezelő VI-ok használata esetén már nekünk kell az egyes műveletek VI-ait elhelyezni a 'Diagram Panel'-en, ám ezért cserébe a fájlműveletek paramétereinek lényegesen finomabb behangolására van lehetőségünk. Ezek a függvények szintén a 'File I/O' palettán találhatók.
A haladó szintű fájlkezeléshez a VI-okat az 'Advanced fájl Functions' alpalettán találjuk meg. Ezekkel az alapelemekkel van lehetőségünk a legkifinomultabb módon megvalósítani az adatok tárolását/kiolvasását egy fájlba/fájlból.

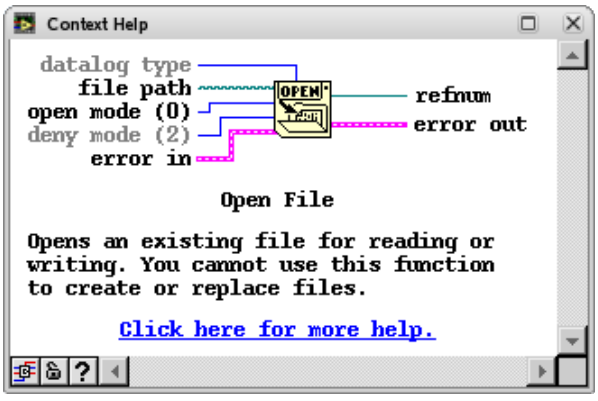
A fájlkezelésnél különösen fontos szerepe van a sorrendiségnek. Ezért különös figyelmet kell fordítanunk ennek megtartására a LabVIEW-ban, ahol alapvetően adatfolyam programozást végzünk. Ez azt jelenti, hogy az adatok párhuzamosan folynak az adatvezetékeken az alkalmazás futtatása során. Azonban az egyes csomópontok (függvények, subVI-ok, struktúrák, stb.)

végrehajtása csak akkor kezdődik meg, amikor minden bemenetükre megérkeztek az adatok. A sorrendiség előírására két struktúra létezik a LabVIEW-ban (III.2. ábra). A különbség közöttük mindössze annyi, hogy míg az egyik a 'Case' struktúrához hasonlóan az egymás után következő eseteket egy összefüggő keretben egymás alatt tárolja ('Stacked Sequence'), addig a másik ennek a kiterített változata, ahol egyszerre láthatjuk az összes 'Frame'-et ('Flat Sequence').



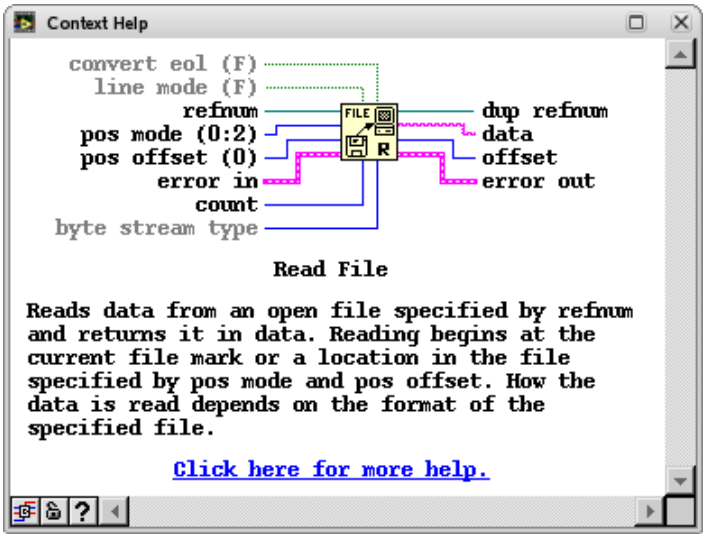
III.2. ábra: Sorrendi struktúrák

Ezután a bevezető után lássunk neki a programunk megvalósításának, mely egy több sort tartalmazó szöveges fájlból fogja kiolvasni a benne tárolt szöveget. (Ehhez készítsünk el egy ilyen fájlt, vagy használjuk a mellékelt „textfile.txt”-t!)
A feladatot a középszintű fájlkezelő VI-okkal oldjuk meg! Ezek sorban a következők: 'Open File' (III.3. ábra) 'Read File' (III.4. ábra) és a 'Close File' (III.5. ábra).



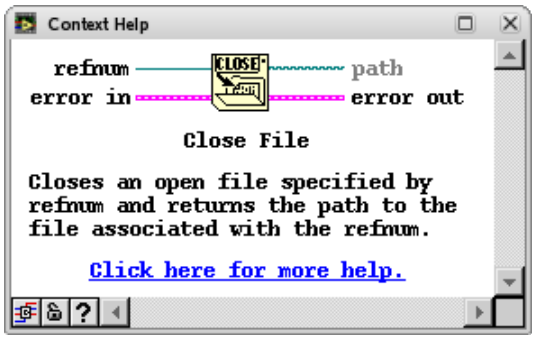
III.3. ábra: Fájl megnyitása

Mint látható ezeknek a VI-oknak igen sok bemenetük van. Az egyes bemenetekről bővebben a 'Click here for more help' feliratra kattintva a LabVIEW súgójából tájékozódhatunk.
Azok a bemenetek, amelyek szürke színnel vannak jelölve, azok opcionális bemenetek. A 'Context Help' ablak bal alsó sarkában levő kis ikonra kattintva kapcsolhatjuk be vagy ki ezek megjelenítését.



III.4. ábra: Fájl olvasása

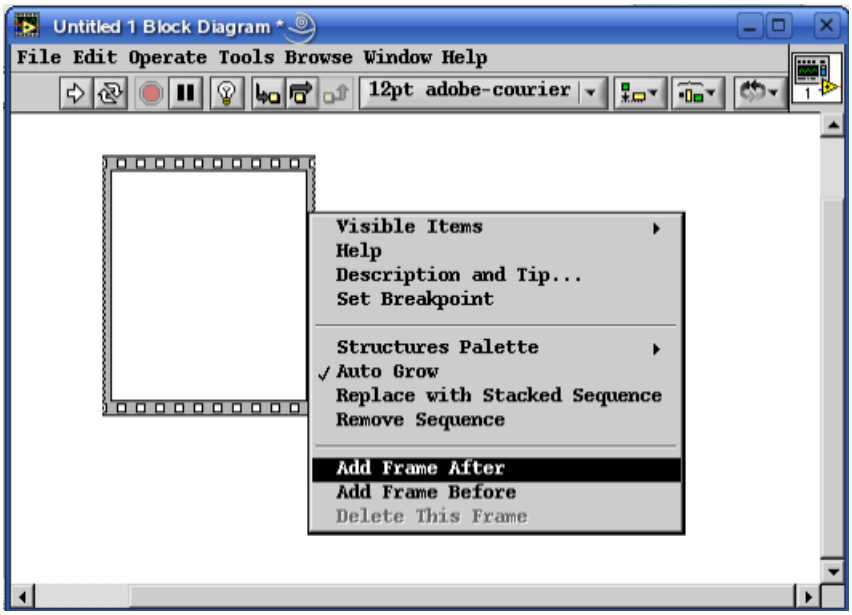
Amennyiben valamelyik bemenet mellett zárójelben egy értéket látunk, az azt jelenti, hogy az ott lévő érték, a VI bemenetén mint bemeneti érték fog megjelenni, ha nem kötünk be adatvezetéket az adott csatlakozóba.
Vannak emellett kötelező bemenetek is, melyeket félkövér betűtípussal láthatunk a 'Context Help' ablakban (hacsak nem úgy, mint példák során használt rendszerben, félkövér az alapértelmezett betűtípus). Ezekbe a csatlakozókba kötelező adatvezetéket csatlakoztatni, különben hibát fog jelezni a rendszer.



III.5. ábra: Fájl bezárása

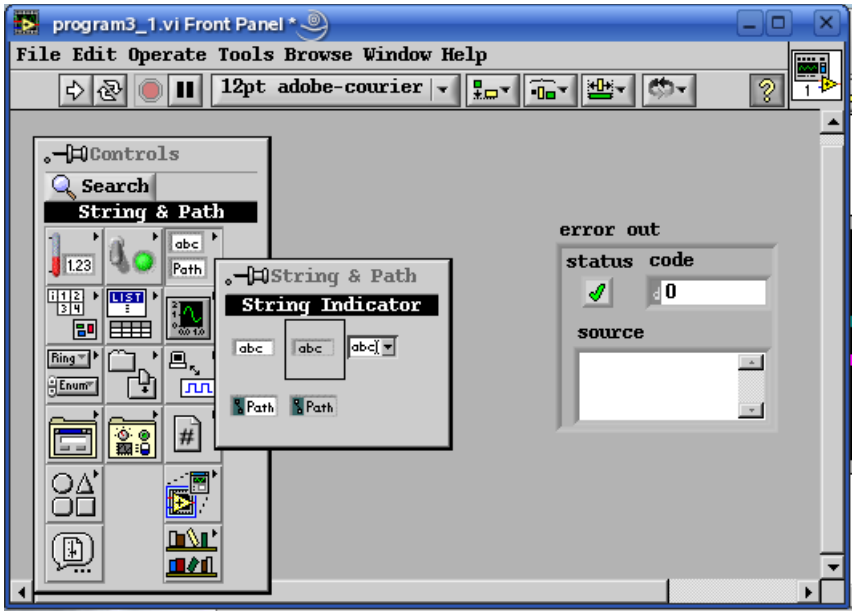
Kezdjük a program írását a sorrendi struktúra 'Diagram Panel'-re helyezésével (most használjunk 'Flat Sequence'-t)! Ez először csak egy keretet tartalmaz, de a hozzátartozó legördülő menü segítségével újabbakat hozhatunk létre (III.6. ábra). Ha megvan a két 'Frame'-ből álló struktúránk, akkor az első keretbe helyezzük el a 'Read File' VI-t, a másodikba pedig a 'Close File' VI-t. Az 'Open File' VI-t a struktúra elé tegyük le.
Egy megnyitott fájlra történő hivatkozás, a fájl kezelése, az 'Open File' függvény 'refnum' kimenetén megjelenő referencián keresztül történik.
Fájlkezeléskor lehetőségünk van egy egységes adattípuson keresztül, az úgynevezett 'Error Cluster'-en keresztül kezelni az egyes műveletek közben esetlegesen felmerülő hibákat.
Mivel a 'Cluster'-ekről még nem esett szó ezért ismerkedjünk most meg az alapvető tulajdonságaival. A 'Cluster' egy olyan elem, amelybe valamilyen logika vagy funkcionalitás vagy egyszerűen az adatvezetékek számának csökkentése végett különböző típusú adatokat

foglalhatunk össze egy egységbe. Az imént említett 'Error Cluster' például egy szöveges ('String'), egy Boolean és egy egész típusú adatot tartalmaz. A 'Cluster'-eket a 'Bundle' és az 'Unbundle' műveleteken keresztül tudjuk összefűzni illetve szétszedni (ezekről bővebben a későbbi fejezetekben lesz szó).
A hibakezelés megvalósítására szolgálnak a fájlkezelő VI-ok 'Error In' illetve 'Error Out' csatlakozói. Kössük tehát össze ezeket ki- és bemeneteket ahogyan az a III.9. ábrán is látszik. Az 'Open File' 'Error In csatlakozóján egy konstanst, míg a 'Close File' 'Error Out' csatlakozóján egy kijelzőt hozzunk létre!



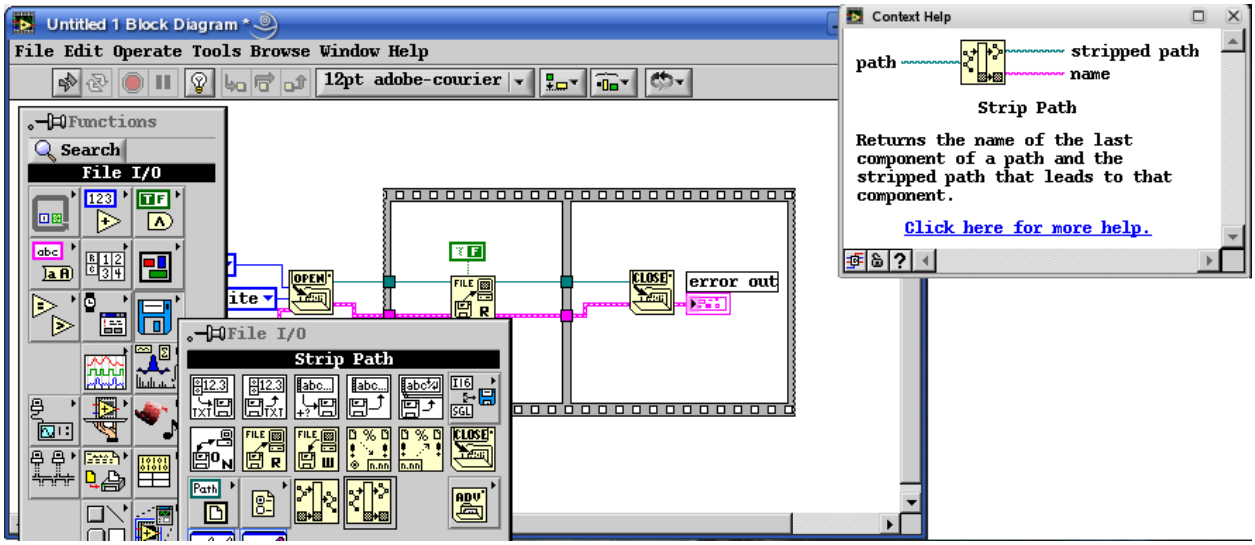
III.6. ábra: Keret hozzáadása a sorrendi struktúrához

Természetesen a beolvasott adatokat meg kell valahol jelenítenünk. Erre pedig a 'String Indicator'-t fogjuk használni, amely a III.7. ábrán mutatott helyen található meg.



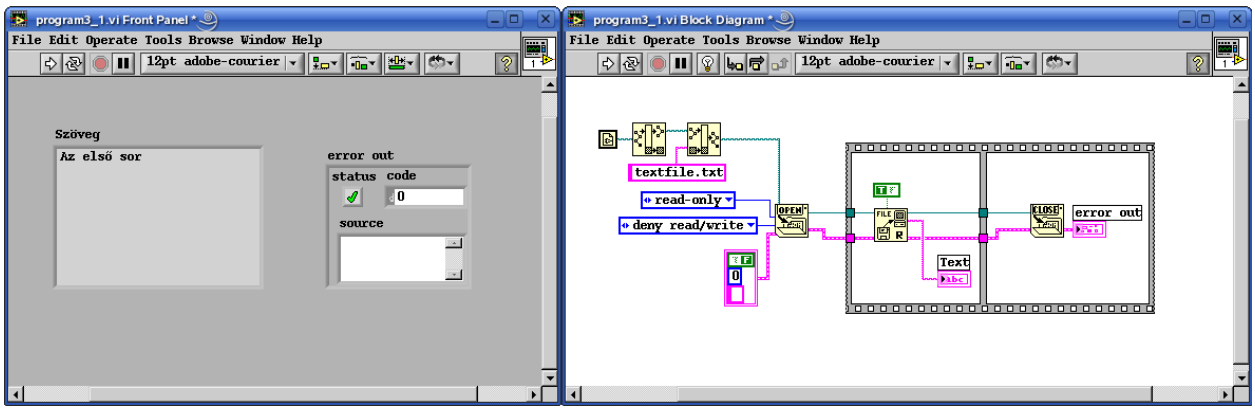
III.7. ábra: A sztring paletta ('Front Panel')

Ez a kijelző, bár amikor elhelyezzük a 'Front Panel'-en egy soros kijelzőnek látszik, valójában nem az (kivéve, ha a legördülő menüjén keresztül előírjuk neki). Mivel mi egy több soros szöveget akarunk kiíratni, ezért húzzuk szét akkorára, hogy a fájlban tárolt szöveg elférjen benne. Amennyiben nem tudjuk biztosan hogy ez így lesz, akkor célszerű a legördülő menüben a 'Visible Items' menüpontból kiválasztani a 'Scrollbar' pontot, ami által görgethető lesz a szöveg, ha kicsinek bizonyult volna megjelenítő. Ezt a kijelzőt a 'Read File' VI 'data' kimenetébe kössük be!
A fájlolvasás VI-ának még egy bemenetére, nevezetesen a 'line mode'-ra készítsünk egy igaz értékű konstanst, mivel nem karakterenként, hanem inkább soronként olvassuk be az adatokat.



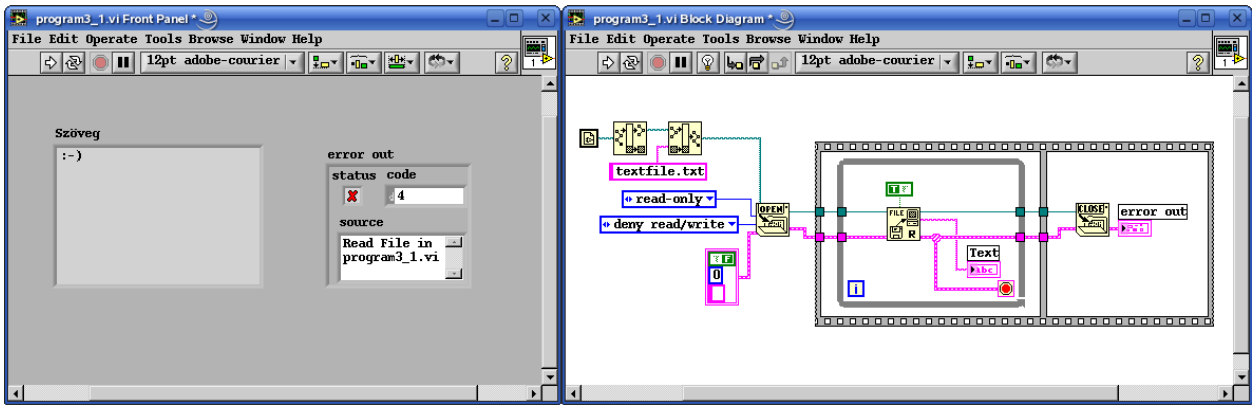
III.8. ábra: Fájlnév és elérési út szétválasztása

Most nézzük, hogy milyen bemeneteknek kell még értéket adnunk a fájl megnyitásáért felelős VI-on. Az 'open mode' és a 'deny mode' bemenetekre készítsünk egy-egy konstanst és állítsuk be őket a III.9. ábrának megfelelően.
Van még egy kötelező bemenetünk, a 'file path'. Ide a megnyitandó fájl elérési útját kell bekötnünk. Egy fájl megadására a LabVIEW-ban a 'path' típusú adat szolgál. Ha ide most konstansként kötnénk be a megfelelő adatot, akkor döntenünk kellene, hogy melyik operáció rendszer alatt akarjuk használni a programot, hiszen a könyvtárszintek elválasztására a Windowsban a 'backslash' (\), míg Mac OS-ban és a Unix alapú rendszereknél a 'slash' (/) szolgál.

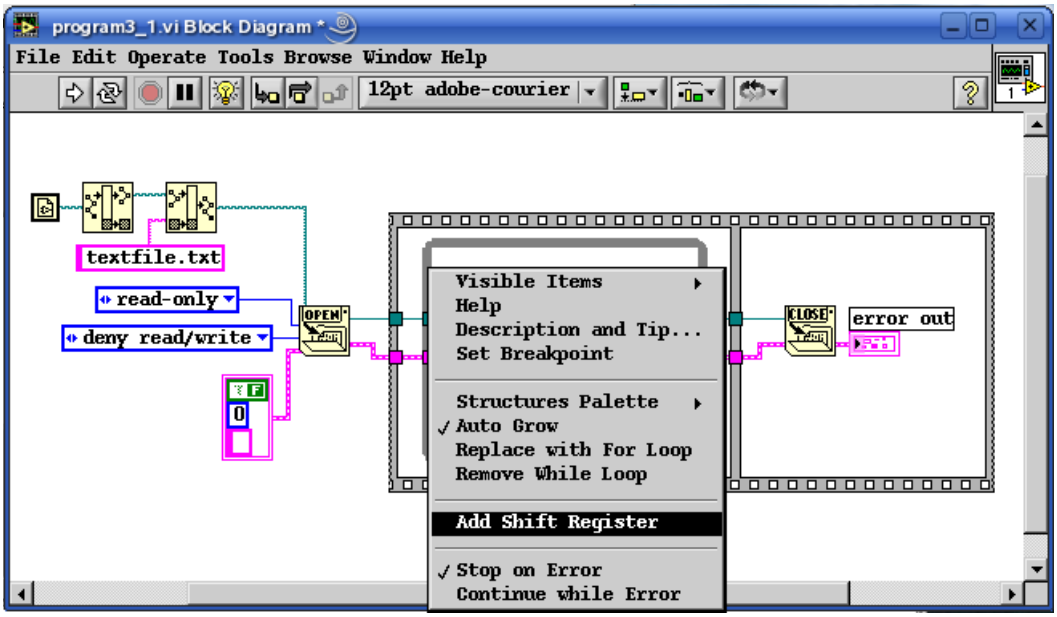


III.9. ábra: A program (verzió 1)

De ha a programunk, és a megnyitandó fájl egy könyvtárban vannak elhelyezve, akkor tudunk készíteni egy egyszerű platformfüggetlen megoldást.
Ehhez keressük meg a 'File I/O' palettán a 'File Constants' alpalettán a 'Current VI's Path' VI-t. Ez a függvény az őt tartalmazó VI abszolút elérési útját adja vissza. Egyetlen probléma van csak vele, hogy ebbe beletartozik magának a VI-nak a neve is. Ezt a 'File I/O' paletta 'Strip Path' elemével orvosolhatjuk, mely szétválasztja az elérési utat és a fájl nevét. E VI édestestvérével a 'Build Path' nevűvel pedig hozzáfűzhetjük a megnyitni kívánt szöveges fájl nevét az elérési úthoz.
Ezzel már egy futtatható programot kaptunk. Nézzük meg, hogy mit ad futási eredményként! Ahogy az a III.9. ábrán is látszik, a program csak a szöveg első sorát olvasta be.
Hogy ezt orvosoljuk, tegyük bele egy 'While' ciklusba a fájlt olvasó programrészt. A ciklus leállítására használhatjuk az 'Error Cluster'-t, hiszen ha nem tud több sort olvasni, akkor hibát fog jelezni, miáltal a ciklus leáll.
A megvalósítás, és az új program futtatási eredménye a III.10. ábrán látható. Sajnos ebben az esetben a fájl utolsó sora látható csak a kijelzőn (igazából minden sor be lett olvasva a fájlból és ki is lett írva, de minden újabb sor kiírása törölte a kijelzőt). A ciklus leállításához használt hiba leírása és kódja pedig az 'error out' kijelzőn olvasható.

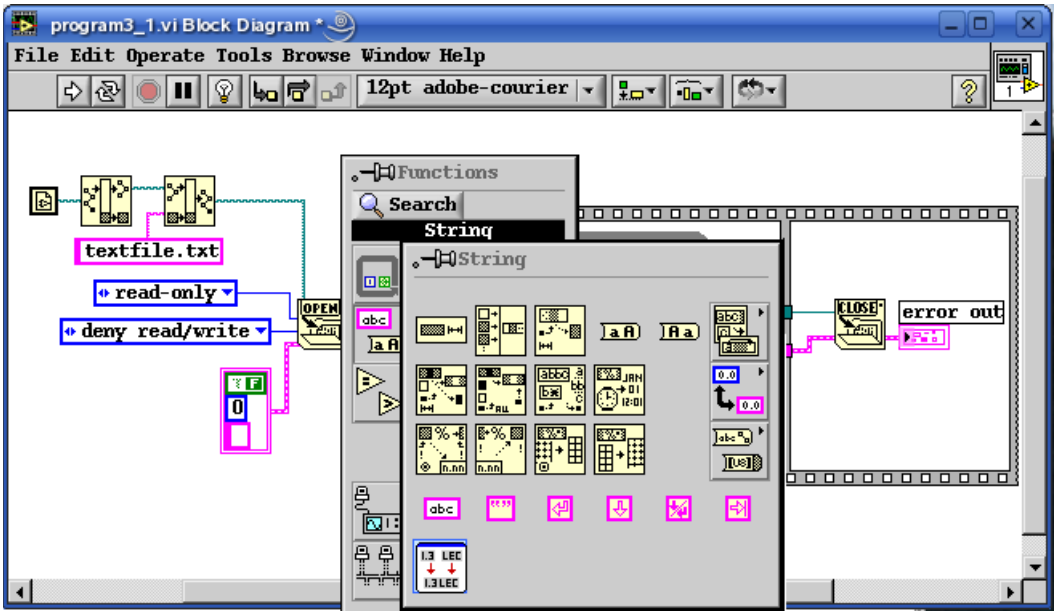


III.10. ábra: A program (verzió 2)



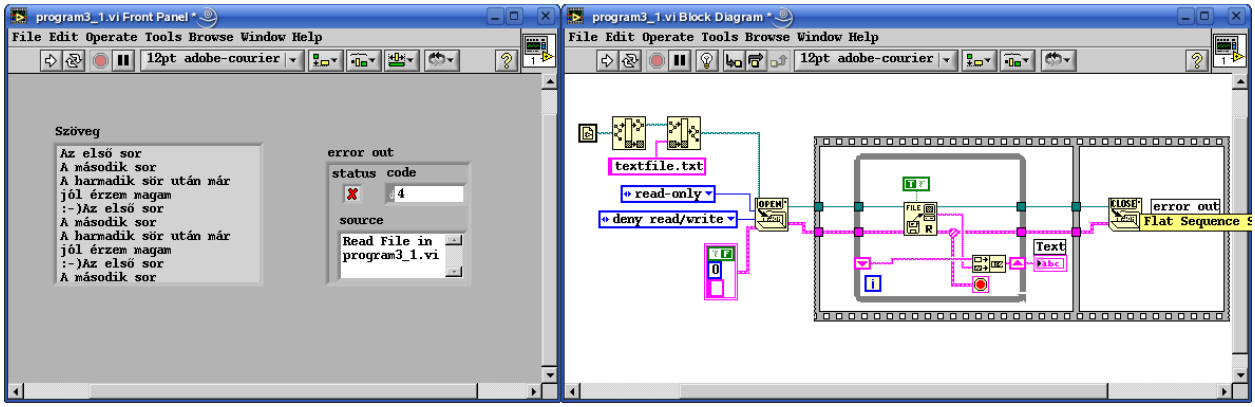
III.11. ábra: 'Shift register' létrehozása

Ahhoz, hogy minden sor meg is maradjon a kijelzőn, egy újabb nagyon fontos elemmel kell megismerkednünk. A 'Shift Register' egy olyan elem, amely lehetővé teszi számunkra, hogy egy ciklusban valamely adatfolyam eredményét a következő ciklus számára elérhetővé tegyük. Jelen esetben a már beolvasott sorokat adja vissza a következő sor beolvasásakor.
'Shift Register'-t a ciklus keretén jobb egérgombbal kattintva, a legördülő menüből adhatunk a ciklushoz (III.11. ábra).



III.12. ábra: A sztring paletta ('Diagram Panel')

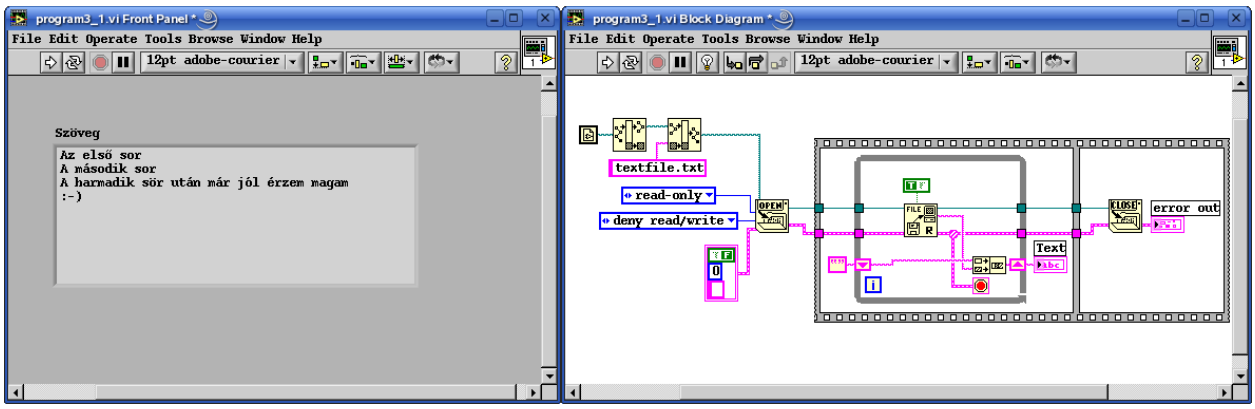
Ahhoz, hogy a beolvasott sorokat összefűzzük, szükségünk van a 'Functions' paletta 'String' alpalettáján található 'Concatenate Strings' VI-ra, amely segítségével az egyes sorokat egymás után fűzhetjük. Mivel a sztringekkel a továbbiakban nem foglalkozunk, ezért az olvasóra van bízva a további sztringfüggvények megismerése.
A programnak most a III.13. ábrához hasonlóan kell kinéznie. Futtassuk le a programunkat néhányszor egymás után!



III.13. ábra: A program (verzió 3)

Azt tapasztaljuk, hogy most már minden sort beolvas a program, ámde az előző futtatások eredménye is benne marad a kijelzőben. Amiből az következik, hogy a 'Shift Regiszter' a program lefutása után is megőrzi a benne tárolt adatokat!

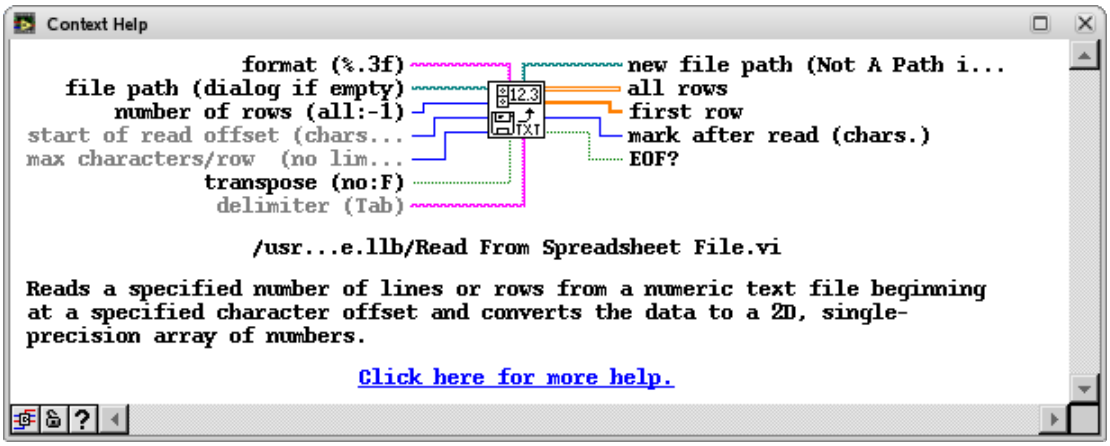
Mivel most ez nem kívánatos, ezért kössünk be a 'Shift Register'-be egy 'Empty String' konstanst a 'String' palettáról, miáltal azt érjük el, hogy a ciklus elindulásakor ezzel a „üres” kezdeti értékkel látjuk el a 'Shift Register'-t.
Mivel az 'Error Cluster' egy átlagos felhasználónak nem jelent információt, ezért a 'Block Diagram'-on az 'error out' kijelző legördülő menüjéből válasszuk ki a 'Hide Indicator' pontot, ami a 'Front Panel'-en elrejti a kijelzőt. A végleges programunk a III.14. ábrán látható.



III.14. ábra: A program (verzió 4)

6. Példa: Olvasás tabulátorral szeparált adatfájlból (program3_2.vi)

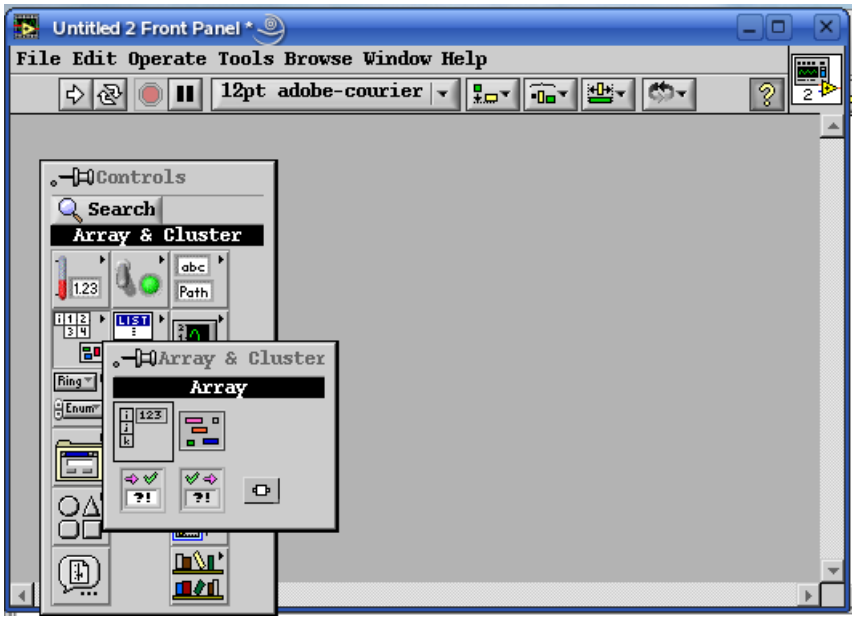
Ebben a feladatban egy magasszintű fájlkezelő VI segítségével fogunk adatokat olvasni egy tabulátorral szeparált adatokat tartalmazó szövegfájlból. Ehhez készítsünk egy ilyen fájlt egy egyszerű szövegszerkesztővel, vagy bármilyen táblázatkezelővel, vagy használjuk a mellékelt „table.txt” fájlt!



III.15. ábra: A 'Spreadsheet' fájl olvasó VI

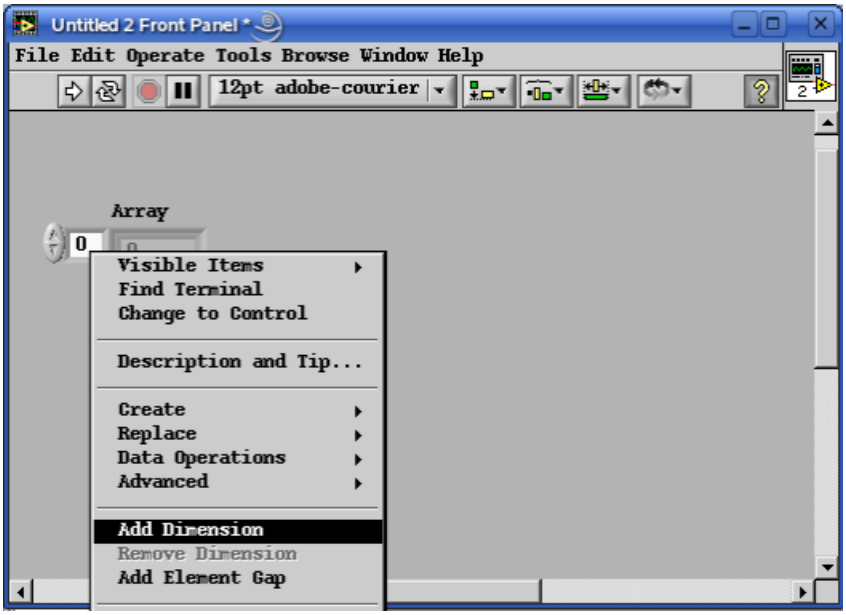
Ez a VI önmaga elvégzi az összes szükséges fájlműveletet, és az 'all rows' kimenetén szolgáltatja a beolvasott két dimenziós tömböt vagyis mátrixot.
A mátrix 'Front Panel'-en való megjelenítéséhez tömb típusú elemet használunk, amely az 'Array and Cluster' alpalettán található (III.16. ábra). Amikor letesszük, még nem tartalmaz semmilyen

elemet sem. A 'Diagram Panel'-en fekete színnel jelenik meg, ami azt jelenti amit a lokális változónál már megismertünk, hogy az adott elem még határozatlan típusú. Hogy egy tömbnek definiáljuk a típusát, ahhoz a 'Front Panel'-en bele kell helyezni egy elemet. Jelen esetben tegyünk bele egy 'Numeric Indicator'-t!



III.16. ábra: A tömb elem helye a 'Controls' palettán

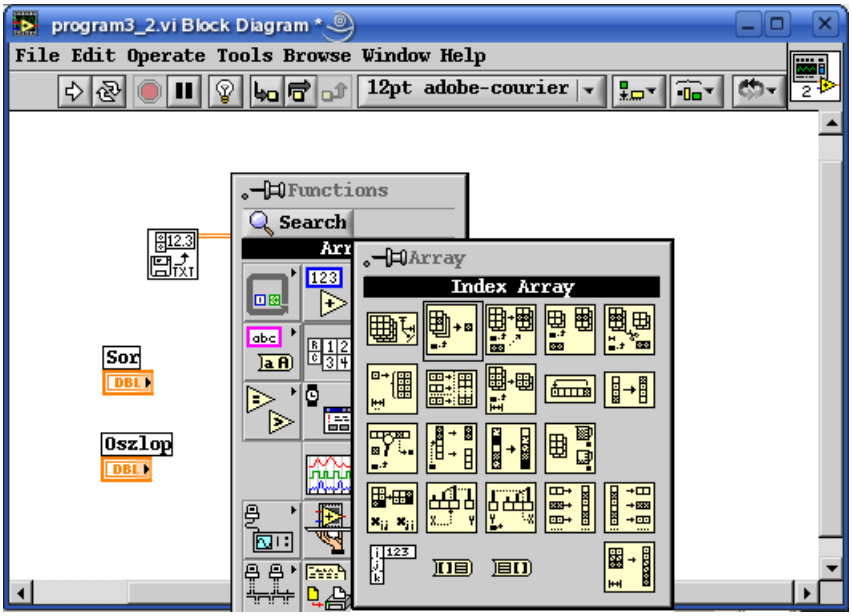
Mivel két dimenziós tömbre van szükségünk, próbáljuk meg széthúzni a tömbünket. Tapasztalni fogjuk, hogy ez csak egy irányban lehetséges. Ahhoz hogy egy tömb dimenzióinak számát megváltoztassuk, a hozzá tartozó legördülő menüből kell kiválasztani az 'Add Dimension' vagy a 'Remove Dimension' menüpontok egyikét (III.17. ábra).



III.17. ábra: A tömb dimenziószámának növelése

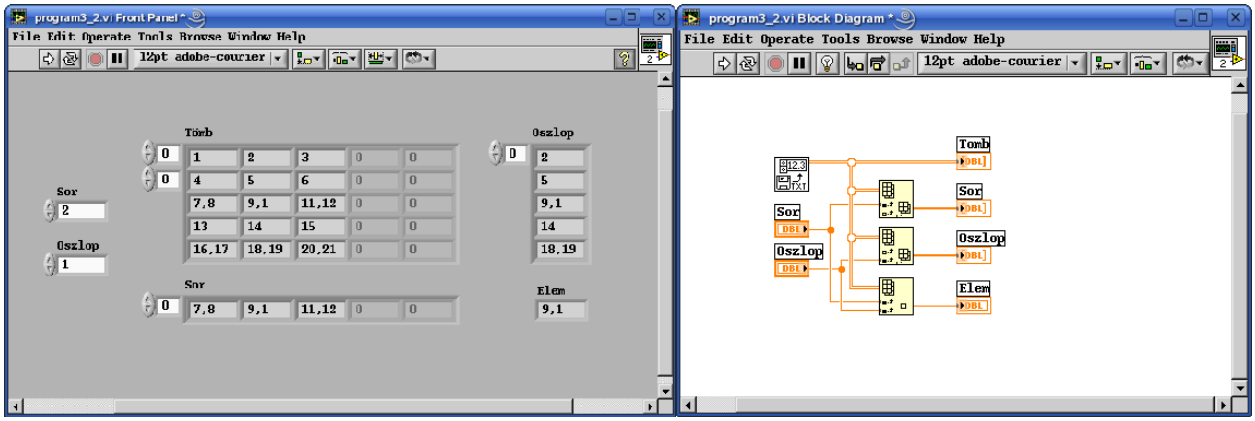
A mátrixunkon kívül jelenítsük meg egy tetszőleges sorát illetve oszlopát, valamint az ezek által

meghatározott elemet! Ehhez helyezzünk még két egy dimenziós tömböt, egy numerikus kijelzőt, valamint a sor illetve oszlop megadásához egy-egy numerikus kontroll el a 'Front Panel'-en, ahogyan az a III.19. ábrán látható.
Egy tömb egy sorát vagy oszlopát, illetve elemét az 'Index Array' VI használatával tudunk kijelölni. Ez több más vektorfüggvénnyel egyetemben az 'Array' palettán található (III.18. ábra).



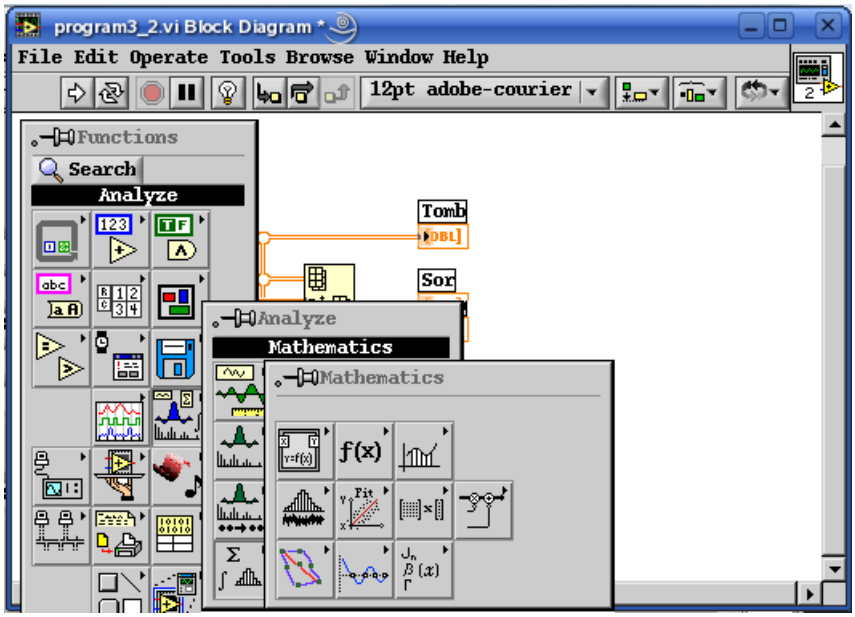
III.18. ábra: A tömbök kezelésére szolgáló VI-ok

Az 'Index Array' VI-nak két bemenete látszik. Amennyiben azonban egy 2 dimenziós tömböt kötünk az 'array' nevű bemenetére, rögtön módosul három bemenetűre. Az egy az 'index (row)' nevet kapja, a másik a 'disabled index (col)' nevet. Ha a sort kijelölő kontrollunkat bekötjük az 'Index Array' 'index (row)' bemenetére, akkor egy 1 dimenziós tömböt, méghozzá a kijelölt sort fogja eredményül adni. Ha a másik bemenetbe az oszlopot jelző kontrollt kötjük, akkor egyrészt a sorokhoz tartozó bemenet fogja a 'disabled'-letiltott jelzőt megkapni, illetve az üres négyszög szimbólumot, másrészt a kimeneten a megfelelő oszlop fog megjelenni. Ha mindkét indexet bekötjük, akkor a sor illetve oszlop index által meghatározott skalár elem lesz a kimenet. A III.19. ábrán látható ennek a megvalósítása.



III.19. ábra: A kész program

Ha lefuttatjuk a programot azt tapasztaljuk, hogy első ránézésre nem a megfelelő, általunk kijelölt sort és oszlopot választotta ki a program. Jusson azonban eszünkbe, hogy a ciklusok ciklusváltozójánál már tapasztaltunk valami hasonlót. A LabVIEW-ban mindenféle indexelési funkció 0-tól kezdődik!
Fontos még megemlíteni, hogy további matematikai illetve mátrixfüggvényeket találhatunk az 'Analyze' paletta 'Mathematics' alpalettáján (III.20. ábra)!



III.20. ábra: További vektor- és mátrixfüggvények helye