PÉCSI TUDOMÁNYEGYETEM

POLLACK MIHÁLY MŰSZAKI FŐISKOLAI KAR

MŰSZAKI INFORMATIKA TANSZÉK

Készült az Európai Unió és a PTE támogatásával.

Sipeky Attila

GRAFIKUS PROGRAMOZÁS LABVIEW-BAN

Főiskolai jegyzet

1. Bevezetés

A LabVIEW (Laboratory Virtual Instrument Engeneering Workbench) egy olyan programcsomag, mellyel virtuális műszereket (virtual instruments) hozhatunk létre. Felhasználható általános célú grafikus programozási nyelvként is, de legnagyobb erőssége, és kifejlesztésének célja a virtuális műszerek tervezése, kialakítása, és az alkalmazások műszerként való használata.

A hagyományos műszer olyan célberendezés, mely adott célra készül, adott bemeneti és kiviteli lehetőségekkel rendelkezik, a felhasználó számára korlátozott, célspecifikus beavatkozási lehetőséget biztosít a minimális kezelőfelületével, és az eredmény megjelenítésére szolgáló felület is csak a szükséges minimum tudással rendelkezik. A műszer belsejében található egységek mikroprocesszor, memória, buszrendszer is az adott feladat végrehajtására készültek. A speciális feladatokra készített műszerek rendszerint nem olcsók, és különböző műveletekhez külön műszereket kell beszereznünk.

A személyi számítógépek mai teljesítménye lehetővé tette, hogy ezeket a speciális eszközöket létrehozzuk virtuálisan egy általános célú PC segítségével. Kialakíthatjuk műszereinket, számítógépünket a speciális illesztőkártyák segítségével (GPIB, VXI, stb.) a technológiához kapcsolhatjuk, és már használhatjuk is a különféle célokra kifejlesztett műszereket.

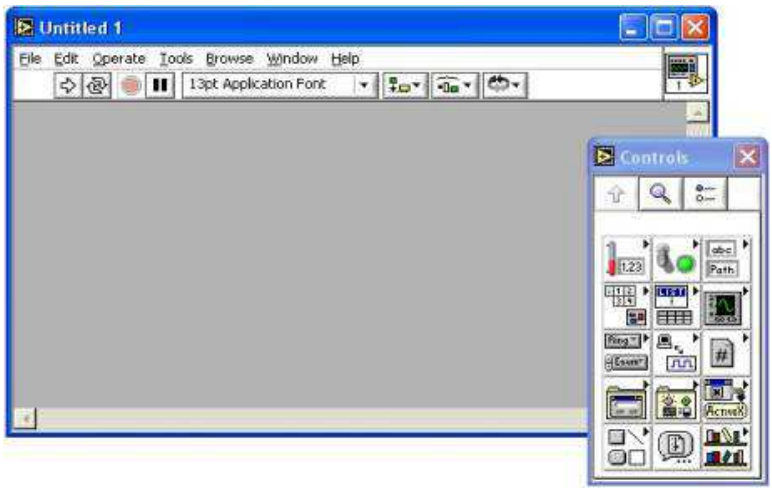
Ehhez a fejlesztési munkához nyújt segítséget számunkra a LabVIEW grafikus programfejlesztő rendszer. A programozás könnyű, a menürendszer nem bonyolult. Könnyedén megtalálhatjuk, és felismerhetjük a keresett elemeket, mivel azok grafikus képét is láthatjuk ikon formájában a kiválasztáskor. A programnyelv mérésadatgyűjtésre specializált elemeket tartalmaz, mint például gombok, kapcsolók, analóg és digitális kijelzők, grafikonok, stb.. Ezek segítségével könnyedén kialakíthatjuk programunk felhasználói felületét, nem kell a bonyolult grafikus elemek képének megprogramozásával bajlódnunk. Ha esetleg mégsem felelnének meg a rendelkezésünkre álló eszközök, kialakíthatunk saját műszereket is, melyek képét bármilyen rajzolóprogrammal elkészíthetjük. Ha kialakítottuk a megfelelő kezelőfelületet, már csak a műszerek közötti kapcsolatokat kell definiálnunk, mely szintén grafikusan történik egy másik ablakban. A LabVIEW programokat virtuális műszernek (virtual instruments) vagy VI-nak nevezik. Mint más programnyelveknél, itt is használhatunk szekvenciát, szelekciókat, az iterációk különböző fajtáit, aritmetikai és logikai műveleteket, függvényeket és készíthetünk alprogramokat (SubVI-okat).

2. A LabVIEW alkalmazásfejlesztő rendszer bemutatása

2.1. ábra. A LabVIEW indulóképe

2.1. Program főbb egységei

Front panel (2.2. ábra): itt alakítjuk ki a felhasználói felületet, melyen elhelyezhetjük a műszereinket. Ezek lehetnek gombok, kapcsolók, grafikonok, be- és kimeneti értékeket kijelző és módosító eszközök. Ezt a felületet látja a felhasználó a program futásakor, itt lehet beleavatkozni a program működésébe, és itt láthatjuk a beavatkozás eredményét is. Ezen ablak háttérszíne alapértelmezés szerint szürke, de ezt a későbbiekben meg is változtathatjuk. A front panelen elhelyezhetünk kontrollokat (bemeneti, vezérlő elemek, melyek befolyásolják a programunk működését) és indikátorokat (kimeneti, kijelző elemek, melyek eredmények megjelenítésére szolgálnak),

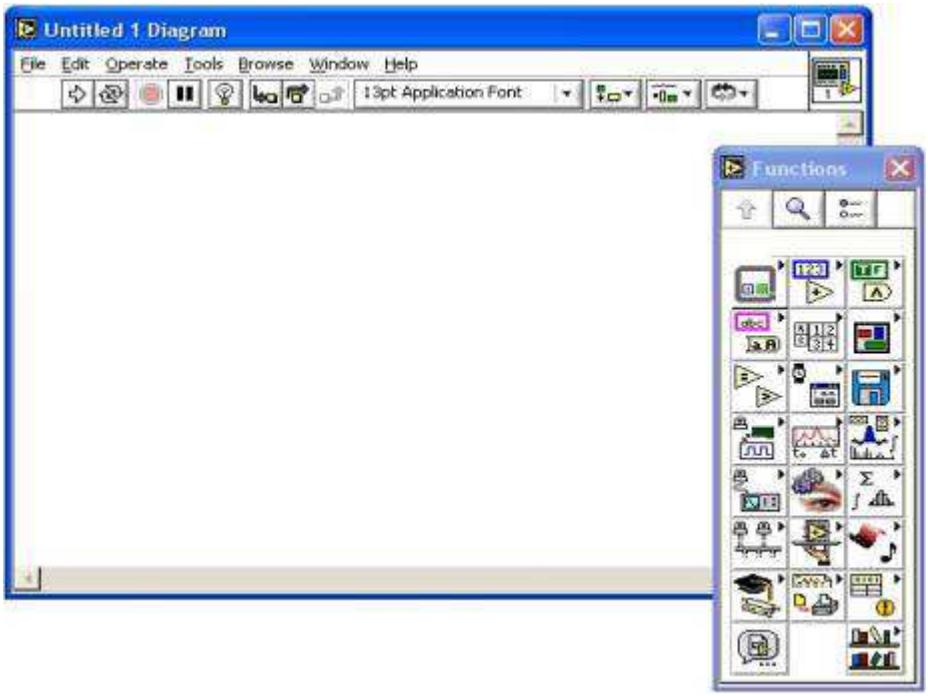


2.2. ábra. Front panel a vezérlők palettájával

melyeket a Vezérlők-palettán (*Controls palette)* érhetünk el (egér jobb gomb lenyomása, vagy Window / Show Controls Palette menüpont). A kontrolloknak és indikátoroknak

különböző opcióik vannak, melyek beállítását az egyes elemek saját pop-up menüjében végezhetjük el.

Diagram panel (2.3. ábra): a programozói felület, ahol a programkészítés grafikus elemek segítségével történik. A block diagramban megjelenik minden elemünk szimbóluma (adattípus szerint), melyeket a front panelen előzőleg elhelyeztünk. Itt kell megteremtenünk az elemek közötti kapcsolatot, beépíteni a programunk működéséhez szükséges struktúrákat, függvényeket, aritmetikai és logikai műveleteket, és minden egyéb szükséges objektumot. Ezeket a Függvények-palettán (*Functions palette)* találjuk (egér jobb gomb lenyomása, vagy Window / Show Functions Palette menüpont). Ezen ablak háttérszíne alapértelmezés szerint fehér, de ezt is megváltoztathatjuk.



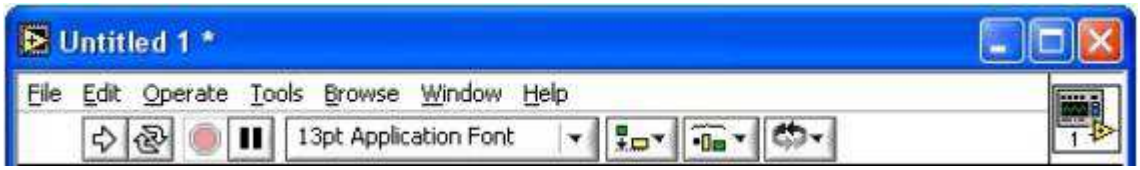
2.3. ábra. Diagram panel a függvények palettájával

Ikon és konnektor (2.4. ábra): Készíthetünk alprogramokat (SubVI), melyeket felhasználhatunk a főprogramban. Ekkor van szükségünk ikon és konnektor használatára, mely mindkét ablak (Front panel és Block diagram) jobb felső sarkában látható. Az ikon adja a programunkhoz tartozó kis képet, a konnektor pedig megmutatja, hogy hova kell kötnünk az alprogram bemeneteit, ill. kimeneteit. Ez az ikon jelenik meg a program block diagram paneljén, mint objektum. Konnektor kialakításával definiáljuk a bemenő és kimenő adatokat.

2.4. ábra. Ikon és konnektor

2.2. Alapvető menüpontok ismertetése

Ebben a fejezetben azokkal a menüpontokkal ismerkedhetünk meg, melyek feltétlenül szükségesek, ill. igen hasznos segítséget nyújtanak már a programozás tanulásának fázisában is. A *2.5. ábrán* látható a menüsor és az ikonsor felépítése!



2.5. ábra. A menüsor és az ikonsor felépítése

A File menü *(2.6. ábra)* 6 egységre osztott, hasonlóan néz ki, mint bármely szerkesztőprogram File menüje. Lehetőségünk van új alkalmazás készítésére (*New VI, New*), már létező megnyitására (*Open*), bezárására (*Close, Close All*), a mentés különböző típusaira (*Save, Save As, Save All*), mely a szokásos menüpontokon kívül tartalmaz speciális lehetőségeket is (*Save with Options, Revert*). Továbbá lehetőségünk van nyomtatásra (*Print, Print Window*), ill. a nyomtatási tulajdonságok beállítására (*Page Setup*). Elvégezhetjük programunk paramétereinek különféle beállításait (*VI Properties*), továbbá megtekinthetjük a legutóbb megnyitott állományok listáját is (*Recently Opened Files*). Végül az *Exit* menüpont választásával kiléphetünk a fejlesztőkörnyezetből.

2.6. ábra. File menü

Az Edit menü *(2.7. ábra)* felső 3 része a szokásos szerkesztési lehetőségeket kínálja (visszavonás, visszavonás visszavonása, kivágás, másolás, beillesztés, törlés, keresés).

A *Costumize Control* menüpont segítségével alakíthatjuk át a már meglévő elemeket, alkothatunk újakat. Az *Import Picture from File* menüpont képfájl beolvasására szolgál, mely kép lehet a programunk díszítőeleme, háttere, vagy - ahogy azzal a későbbiekben részletesebben megismerkedünk - egy-egy elemünk új képe.

Fontos menüpont a *Remove Broken Wires*, hiszen programunk készítése során gyakran előfordul, hogy felesleges vagy hibás összekötések alakulnak ki, ezeket e menüpont

segítségével egy kattintással el tudjuk tüntetni. Ha ezt nem tesszük meg, a programunk nem futtatható.

A *Run-Time Menu* lehetővé teszi, hogy saját menüt készítsünk.

2.7. ábra. Edit menü

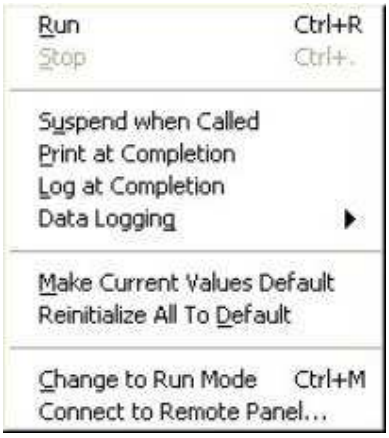
A Operate menü *(2.8. ábra)* első két pontja a program futtatása (*Run*) és leállítása (*Stop*), melyet az ikonsorról is megtehetünk.

A *Suspend when Called* menüponttal beállíthatjuk, hogy a program megszakítsa-e működését, ha alprogramként hívjuk meg.

A *Print at Comletion* és a *Log at Comletion* menüpontokkal kinyomtathatjuk, vagy eltárolhatjuk a programunk lefutása utáni végeredményt.

A harmadik szakaszban, beállíthatjuk, vagy visszaállíthatjuk programunk elemeinek alapértelmezett értékeit.

A *Change to Run Mode* vagy *Change to Edit Mode* ponttal futtató, vagy szerkesztő módba válthatunk.



2.8. ábra. Operate menü

A Tools menü *(2.9. ábra)* első menücsoportjában a műszerekkel, és DAQ-val kapcsolatos beállításokat tehetjük meg.

A *VI Revision History* menüpontban megjegyzéseket fűzhetünk a programunkhoz, melynek fő funkciója, hogy a módosításokat dokumentálhassuk.

A harmadik csoportban a programkönyvtárak kezelését segítő menüpontok találhatók. Összetartozó programjainkat program-könyvtárakba (Library) rendezhetjük, így biztosan egy helyen fogjuk őket tárolni, ill. megtalálni.

Az *Advanced* menüpontban import, export lehetőségeket találunk. Az *Options* menüpontban pedig a programunkra vonatkozó beállításokat végezhetjük el.

2.9. ábra. Tools menü

A Browse menüben *(2.10. ábra)* programunk felépítését, szerkezetét, az alprogramokat, az általunk készített elemeket és a töréspontokat tekinthetjük át.

2.10. ábra. Browse menü

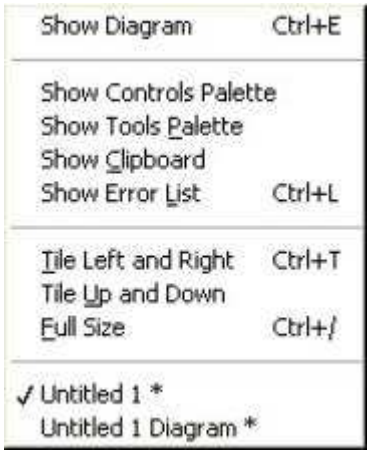
A Windows menü *(2.11. ábra)* igen fontos része a programnak. Már esett arról szó, hogy a programunk szerkesztésekor egyszerre két ablakot kell használnunk, a Front panelt és a Diagram panelt. Ha a Front panelt látjuk, akkor a Windows menü legfelső menüpontja a *Show Diagram*, mellyel átválthatunk a Diagram panelre, ha a Diagram panelen dolgozunk, akkor a *Show Panel* felirat jelenik meg, mellyel a Front panelre válthatunk.

Fontos menüpont a *Show Controls Palette* menüpont, mely Front panel esetén jelenik meg, és a felhasználói felületen elhelyezhető elemeket tartalmazza adattípusonként csoportosítva. Diagram panel esetén a *Show Functions Palette* menüpont jelenik meg, ahol szintén adattípusonként csoportosítva az elemekkel végezhető műveleteket találjuk. A palettákat közvetlenül is előhívhatjuk, ha az adott panelen az egér jobb gombjával kattintunk.

A következő menüpont a *Show Tools Palette*, mely a program kialakítását segítő eszközöket tartalmazza, melyek feladatairól a későbbiekben esik szó.

Megjeleníthetjük továbbá a vágólapot és az elkövetett hibákra figyelmeztető listát. Itt rendezhetjük a megnyitott ablakainkat egymás mellé, ill. egymás alá, vagy állíthatjuk azokat teljes képernyő méretűre.

A Windows menü legalsó részében a megnyitott ablakok listáját találhatjuk, ahol ki is választhatjuk, melyik legyen előtérben, melyik legyen az aktuális.



2.11. ábra. Windows menü

A Help menü *(2.12. ábra)* első menüpontja nagy segítséget nyújthat a programkészítés folyamatában. A *Show Context Help* menüpont egy kis ablakot nyit meg, melyben az éppen használatos elemről vagy műveletről kaphatunk hasznos segítséget. Ennek a Help ablaknak rögzíthetjük a tartalmát a *Lock Context Help* menüponttal.

A *VI, Function, & How-To Help* menüpont a hagyományos Help, ahol akár címmszóra kereshetünk, de böngészhetünk a tartalomjegyzékben is. Az alatta lévő menüpont (*Search the LabVIEW Bookshelf*) hasonló funkciójú az előzőhöz, csak elektronikus könyv formájában tárja elénk az információkat. A többi menüpont a speciális lehetőségekkel kapcsolatos segítséget, és példák bemutatását nyújtja számunkra.

2.12. ábra. Help menü

2.3. Elkészített alkalmazás futtatása

A futtatás ikonjait a menüsor alatti eszközsor *(2.5. ábra)* elején találjuk:

 *Futtatás gomb (Run)* : Ha működőképes a program, azaz futtatható, akkor ennek a gombnak a megnyomásával indíthatjuk el a programunkat.

 Ha az alkalmazás nem futtatható, akkor a nyíl törött képet mutat.

 A Run gomb képe megváltozik a futás közben.

 A Run gomb képében egy kis nyilacska van futás közben, ha az alkalmazásunk alprogramként (SubVI) működik.

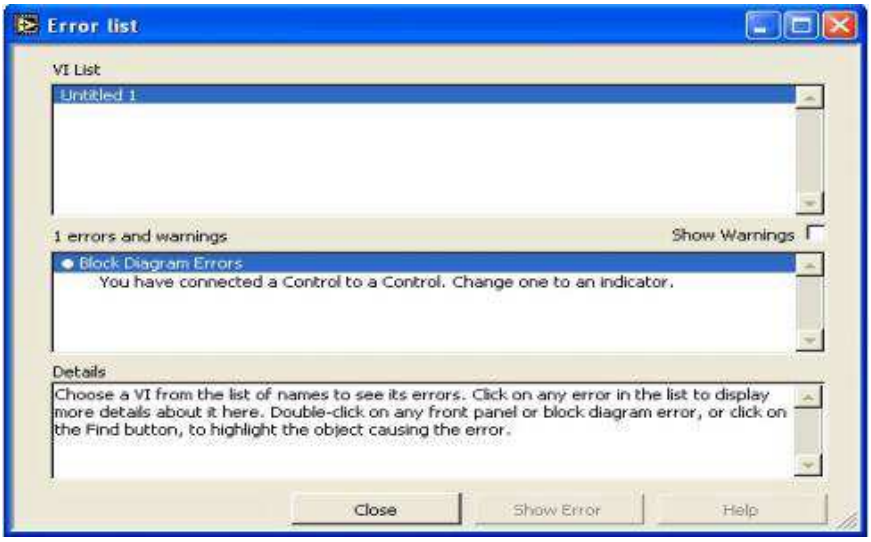
 *Folyamatos futtatás gomb* : a program futtatása folyamatosan ismétlődik.

 *Stop gomb* : leállítja a program futását.


Pause gomb : szünetelteti a program futását. Ahhoz, hogy tovább folytassuk a programot, kattintsunk ismét a gombra.

A Diagram panelen további gombok is találhatók, melyek a programunk tesztelésére szolgálnak. Ezekről később részletesen is szó lesz.

Ha programunk hibás, nem futtatható, akkor is érdemes a futtatás gombra kattintani (törött nyíl), mert ekkor a megjelenő hibalista *(2.13. ábra)* segítségével hamarabb felfedezhetjük a hiba okát.



2.13. ábra. Hibalista

2.4. Elemek tulajdonságai

Az elemek Front panelre való helyezésére két lehetőségünk van. Vagy a Front panel Windows menüjéből kiválasztjuk a *Show Controls Palette* menüpontot, vagy a panelen állva az egérkurzorral lenyomjuk az egér jobb gombját. Ez utóbbi esetben a paletta használat után rögtön eltűnik, kivéve, ha a bal felső sarokban megjelenő rajzszeggel rögzítjük. Mindkét esetben a Controls palettát *(2.14. ábra)* kapjuk eredményül, ahol a felhasználható elemek adattípusonként vannak csoportosítva.

Itt megtalálhatjuk a hagyományos programnyelvekben is fellelhető típusokat, mint például numerikus típus, logikai típus, szöveges típus, tömb, és még egyéb típusokat. Ha az egérkurzorral rámutatunk a különböző típusokra, láthatjuk, hogy egy típuson belül is igen széles választékot kínál a program. Egy elemre rákattintva elhelyezhetjük azt a felhasználói felületen, majd rögtön megadhatjuk az új elemünk nevét.

2.14. ábra. Controls palette

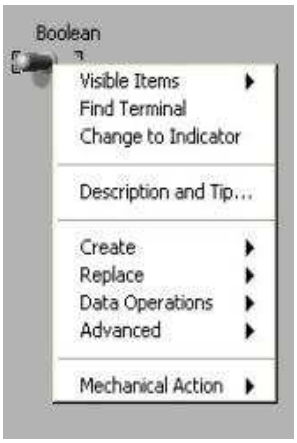
Minden elem rendelkezik bizonyos tulajdonságokkal, melyek vagy a megjelenését, vagy a működését befolyásolják. Egy elem két féle módban lehet, vagy kontrol (control), vagy indikátor (indicator). Ha egy elem kontrol, akkor vezérlőelemként működik, amely segítségével beavatkozhatunk programunk működésébe a futás során. Ha egy elem indikátor, akkor kijelzőként működik, az ilyen kijelzők mutatják programunk működésének eredményeit. Azt, hogy egy elem kontrol vagy indikátor legyen, az elem saját pop-up menüjében *(2.15. ábra)* tudjuk beállítani, melyet az elemre mutatva az egér jobb gombjával hívhatunk elő. Ha az elemünk kontrol, akkor a *Change to Indicator* feliratot láthatjuk, ha pedig indikátor, akkor a *Change to Control* menüponttal válthatunk át.

A *Find Terminal* menüpont megmutatja, hogy hol található az adott elem megfelelő képe a másik ablakban. Ez nagy segítséget jelenthet bonyolult, sok elemet tartalmazó program esetén.

A *Visible Items* menüpont kiválasztásakor újabb legördülő menüt láthatunk, ahol az elemünkhöz tartozó kiegészítő információkat jeleníthetjük meg, vagy rejthetjük el.

A *Replace* menüpont választásával kicserélhetjük a nem kívánt elemünket egy bármilyen más típusú elemre.

A pop-up menü egyéb részeiben található menüpontok az elem működésének beállítását teszik lehetővé. Ez a különböző típusoknál más és más.

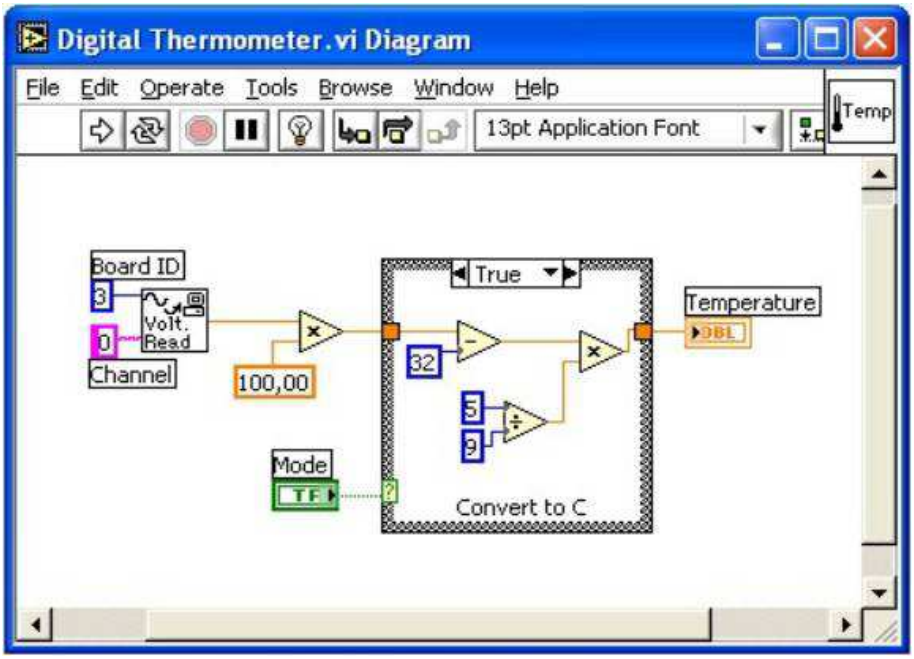


2.15. ábra. Elem saját menüje

3. Programkészítés a LabVIEWban

A tényleges programkészítés a Diagram panelen történik, de még ez előtt ki kell alakítanunk a felhasználói felületet a Front panelen. Itt kell elhelyeznünk az összes vezérlő és megjelenítő elemet (Control, Indicator) és csak ezután lépjünk át a Diagram panelre, ahol megjelennek a vezérlő és megjelenítő elemek megfelelő szimbólumai. Ekkor el kell még helyeznünk a programunkhoz szükséges csomópontokat, és azok megfelelő termináljait összekötve egymással és a Front panelen található elemek szimbólumaival készen is van a program.

A *3.1. ábrán* látható egy elkészített program kódja.



3.1. ábra. Grafikus forráskód LabVIEW-ban

(A példaprogram egy hőmérsékletmérő feszültségjelét olvassa be, és jeleníti meg értékét Celsius fokra átszámítva.)

3.1. Alkalmazás készítéséhez szükséges eszközök

Ahhoz, hogy egy komplett programot tudjunk alkotni, szükségünk van bizonyos segédeszközökre. Ezeket az eszközöket az Eszközök-palettán (*Tools palette*) találhatjuk, melyet Windows menü *Show Tools Palette* menüpontjával érhetünk el. Mielőtt valamilyen műveletet végrehajtunk, ki kell választanunk a célnak megfelelő eszközt az Eszközök-palettáról *(3.2. ábra)*. Az Eszközök-palettán található programszerkesztő eszközök:

3.2. ábra. Eszközök palettája

Az Eszközök-palettán található programszerkesztő eszközök:

 *Operating tool*: mellyel a front panelen található kontrollok és indikátorok értékeit változtathatjuk.

 *Positioning tool*: mellyel az objektumok kiválasztását, mozgatását, és átméretezését végezhetjük el.

 *Labeling tool*: mellyel feliratokat, címkéket helyezhetünk el.

 *Wiring tool*: mellyel megteremthetjük az objektumok közötti összeköttetést a block diagramban.

 *Object Shortcut Menu tool*: mellyel megjeleníthetjük az elem saját menüjét, de ezt megtehetjük az egér jobb gombjának lenyomásával is.

 *Scrolling tool*: mellyel a gördítősávok használata nélkül változtathatjuk az ablak látható területének pozícióját.

 *Breakpoint tool*: mellyel töréspontot helyezhetünk el, vagy törölhetünk a programban.

 *Probe tool*: mellyel mérési pontokat helyezhetünk el, és ezek futás közben megjelenítik az adott helyen az adatok aktuális értékét.

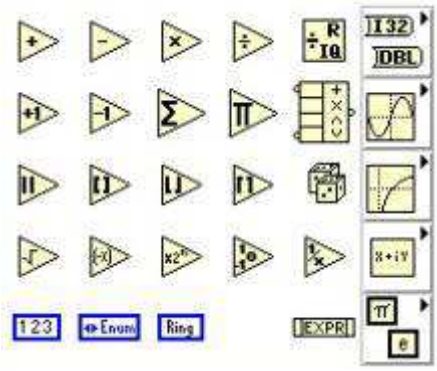
 *Color Copy tool*: mellyel adott helyen lévő aktuális színt másolhatunk át a színező eszközbe (*Coloring tool*).

 *Coloring tool*: mellyel minden felület és objektum színét megváltoztathatjuk. A két négyzet az előtér és a háttér színeit mutatja.

3.2. Nodes (csomópontok)

A csomópontok program végrehajtható elemei. Négy csomóponti elemtípust különböztethetünk meg:

1. A függvények (functions) beépített csomópontok, amelyek az alapvető műveletek végrehajtására szolgálnak. Ezeket elemtípusonként csoportosítva találhatjuk meg, pl.: számokkal végezhető műveletek *(3.3. ábra)*, fájl műveletek vagy szöveg típusú adatok kezelése.

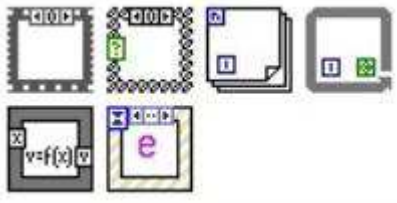


3.3. ábra. Numerikus műveletek

2. Az alprogram (subVI) *(3.4. ábra)* olyan VI, mely egységbe foglal egy előre összeállított műveletsorozatot, és a főprogramból meghívható, mint egy egyszerű művelet. A bemenetek alapján szolgáltat valamilyen kimenetet.

3.4. ábra. Alprogram ikonja

3. A struktúrák (S tructures) *(3.5. ábra)* lehetnek ciklusok, szekvencia, elágazások, képlet vagy esemény.



3.5. ábra. Strukturális műveletek

4. A kód kapcsolati csomópontok (CINs=Code Interface Nodes) *(3.6. ábra)* a felhasználó által készített gépi kódú függvények, melyek C nyelven készülhetnek. Segítségével kiküszöbölhető a LabVIEW azon hiányossága, hogy nem képes megvalósítani a rekurzív hívásokat.

3.6. ábra. Code Interface Node

3.3. Terminals (Kapcsolódási pontok)

Minden elemnek van vagy vannak kapcsolódási pontjai, ahova a bemenő, ill. kimenő adatokat kell kötnünk. A bemenetek valamely vezérlő elemből jövő adatok lehetnek, vagy annak csomópontokon áthaladó, átalakított változatai. Egy csomópont kimenetét megjelenítő elemre köthetjük (eredmény kijelzés céljából), vagy további csomópontok bemenetéül szolgálhat.

Annak érdekében, hogy az elemek megfelelő termináljait kössük össze, egy kis segítséget nyújt az a lehetőség, hogy az elemeknek nem csak az ikonját, hanem a terminálok elhelyezkedését is megjeleníthetjük. Ezt az elem saját pop-up menüjének *Visible Items / Terminals* menüpontjának kiválasztásával tehetjük meg.

3.4. Elemek közötti kapcsolat megteremtése

A LabVIEWban az elemek közötti kapcsolat definiálása szintén grafikusan történik. Az Eszközök-palettából kiválasztjuk a *Wiring tool*-t, és ezzel a "cérnatekerccsel" egyszerűen összekötjük elemeinket *(3.7. ábra)*. Rákattintunk az egyik elemünkre, és kattintunk a bal egérgombbal, majd elmozgatjuk az egérkurzort a második elemünkig és ott is kattintunk. Összekötésnél teljesen mindegy, hogy melyik elemtől kiindulva teremtjük meg a kapcsolatot, viszont fontos, hogy az objektumok megfelelő kapcsolódási pontjait kössük össze. Erről úgy győződhetünk meg, hogy amikor az egér "cérnatekercs" alakú mutatóját az elemünk fölé húzzuk, akkor annak villogni fog az a kapcsolódási pontja, ami fölött a kurzorunk van. Ebben a pillanatban ezt a terminált tudjuk csatlakoztatni.

Ha a csatlakoztatási pont előtt is kattintunk az egér bal gombjával, akkor törést tehetünk az összekötő vezetékbe. A törés irányát a *Space* billentyűvel változtathatjuk meg. A vezeték

megtörésének akkor van nagy jelentősége, ha összetett programot készítettünk, és az átláthatóság kedvéért ki szeretnénk kerülni a már elhelyezett elemeket. Ha minden elemet "bekötöttünk" a programunkban, beleértve a szükséges műveleteket, el is készült az alkalmazásunk.

(a) (b)

3.7. ábra. Elemek összekötése

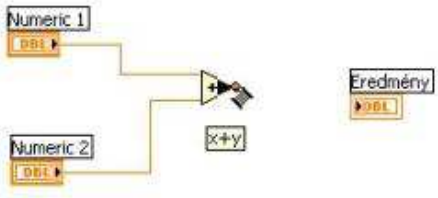
5. Adattípusok jelölése

Ha egy elemet bekötöttünk, akkor a vezeték színe és mintázata megváltozik, a vezetéken áramló adat típusához igazodik. A logikai típus színe a zöld, a lebegőpontos típusoké a narancssárga, az egész típusoké a kék, a szöveges típusé a lila és így tovább, minden típusnak megvan a maga rá jellemző színe *(3.8. ábra)*. A vonal mintázata az adatstruktúrát ábrázolja, például a szimpla vonal egyszerű adatot, a vastag vonal egy dimenziós, a dupla vonal két dimenziós tömböt jelöl. A vastag keretes szimbólum a vezérlő, a vékony keretes a megjelenítő elemet reprezentálja.

3.8. ábra. Típusok megjelenése a forráskódban

3.6. Tip Strip (Csúcs címke)

A csúcs címke *(3.9. ábra)* egy sárga színű szövegmező, amely megmutatja a kapcsolódási pont nevét, ha a cérnatekercs alakú egérkurzort a terminál fölé mozgatjuk. A csúcs címke megkönnyíti a függvények és alprogramok kapcsolódási pontjainak azonosítását az összeköttetés kialakításakor.



3.9. ábra. Csúcs címke

3.7. Property Node (Tulajdonság csomópont)

Tulajdonság csomópontot *(3.10. ábra)* létrehozhatunk bármely elemhez. Segítségével az elemek minden tulajdonságát elérhetjük program futása közben is, nem csak a program futtatása előtt tudjuk beállítani. Létrehozásához a Diagram panelen az elem szimbólumának pop-up menüjében a *Create/Property Node* menüpontot kell választanunk. Egyszerre tetszőleges számú tulajdonságot jeleníthetünk meg. A bővítés az *Add Element* menüponttal történik. A tulajdonság lehet vezérlő, vagy megjelenítő tulajdonságú. Az ábrán látható tulajdonság csomópont egy grafikus megjelenítő elemhez tartozik. Programon belül állíthatjuk akár a tengelyek minimumát, maximumát, a kirajzoló vonal stílusát, a megjelenítő méreteit, és még rengeteg egyéb tulajdonságát is.

3.9. ábra. Tulajdonság csomópont

3.8. Objektumok kezelése

3.8.1. Kijelölés

A pozícionáló eszközzel (*Positioning Tool*) kiválaszthatjuk az objektumokat mind a front, mind pedig a diagram panelon. Az objektum kiválasztásakor a pozícionáló eszközhöz tartozó egérmutatót az objektum fölé mozgatjuk, és az egér bal gombjával kattintunk. Amikor az objektumot kijelöljük, egy mozgó szaggatott vonal keret veszi körül. Ha több mint egy objektumot akarunk kiválasztani, akkor minden további objektumnál nyomjuk le a *Shift* billentyűt és kattintsunk az egér bal gombjával. Több elem kiválasztásakor egy üres területre kattintva lenyomott egér billentyű mellett "bekeretezzük" azokat az objektumokat, amelyeket ki szeretnénk jelölni.

3.8.2. Mozgatás

Miután kijelöltük a mozgatni kívánt elemeket, a bal egérgombbal az egyik kiválasztott objektumra kattintunk, és lenyomott egérgomb mellett egyszerre elmozgatjuk az összes

kiválasztott elemet. Az elemek mozgatására billentyűzettel is lehetőségünk van. Kijelölés után a nyilakkal mozgathatjuk a kijelölt objektumcsoportot.

3.8.3. Másolás

Kiválasztás után az Edit menü Copy, majd Paste menüpontjainak választásával másolhatjuk elemeinket, vagy egérrel ugyanúgy teszünk, mintha mozgatnánk az objektumot, de közben lenyomva tartjuk a Ctrl billentyűt. Alkalmazható továbbá a Windowsban jól megszokott Ctrl+C és Ctrl+V kombináció is. A legelső lehetőséggel nem csak egy programon belül, hanem VI-ok között is másolhatunk.

3.8.4. Törlés

Az objektumok törlése kijelölést követően a *Delete* billentyű lenyomásával történhet, illetve használhatjuk erre a célra az Edit menü *Clear* menüpontját is.

3.8.5. Átméretezés

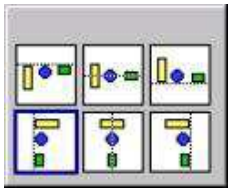
Az objektumok méretét a pozícionáló eszköz segítségével változtathatjuk meg. Válasszuk ki a pozícionáló eszközt, helyezzük a mutatót az objektum egyik sarka fölé úgy, hogy a mutató egy keret sarkának megfelelő alakúvá változzon. Ekkor kattintva az egér bal billentyűvel és lenyomva tartva azt a keret alakú mutatóval az objektumot a kívánt méretűre változtathatjuk. Az objektum új méretét egy szaggatott keret mutatja. Amikor elengedjük az egér billentyűt, az objektum új méretében jelenik meg. Ez az eljárás alkalmazható objektumokra, címkékre, struktúrákra és konstansokra.

3.8.6. Igazítás

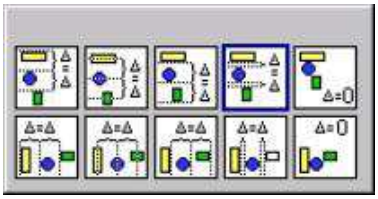
Lehetőségünk van az elemek egymáshoz viszonyított igazítására, és azonos távolságra való elosztására az eszközsor elemeinek segítségével, melyek közül az elemek kijelölése után választhatunk.

Az első ikont választva (*Align Objects*) *(3.11. (a) ábra)* és a legördítő nyílra kattintva az elemek igazítására vonatkozó, 6 ábrát tartalmazó ikonmenü jelenik meg. Itt választhatjuk ki, hogy az elemeket milyen irányba rendezzük.

A második ikon (*Distribute Objects*) *(3.11. (b) ábra)* kiválasztásával az objektumok egymás közötti távolságát tehetjük arányossá különböző szempontok szerint.



(a)



(b)

3.11. ábra. Elemek igazítása

3.8.7. Egymás fölé helyezés

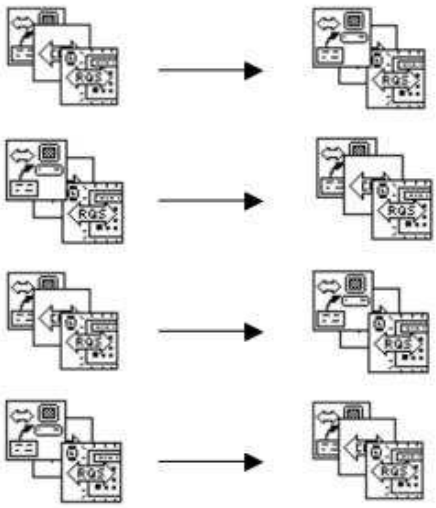
Az elemek egymásra helyezhetők, és mindig az utolsóként elhelyezett elem látszik teljes egészében. Ha ezt a sorrendet meg kívánjuk változtatni, akkor az eszközsor *Reorder*  ikonja alatt található menüpontokat kell használnunk. Az elemek lehetséges elhelyezkedése látható a *3.12. ábrán.*

Ha például van három elemünk, és a legelőször elhelyezettet szeretnénk legfelül látni, akkor az első elem kijelölése után a *Move To Front* menüpontot kell választanunk.

Ennek pont az ellenkezőjét teszi a *Move To Back* menüpont, az elöl lévő elemet leghátra helyezi.

A *Move Forward* menüpont eredménye, hogy a takarásban lévő elemünk egyel előrébb vándorol.

A *Move Backward* menüpont az előző ellenkezőjét végzi, a kijelölt elemünket egyel hátrébb helyezi.



3.12. ábra. Elemek lehetséges elhelyezkedése

3.9. Ellenőrző kérdések

1. Melyek a programfejlesztő környezet főbb egységei?

2. Mi a funkciója a Front panelnek?

3. Mi a funkciója a Diagram panelnek?

4. Mi a funkciója a Konnektornak?

5. Mire használható a Context Help?

6. Milyen lehetőségek vannak az elkészített alkalmazás futtatására?

7. Mit nevezünk Controlnak?

8. Mit nevezünk Indicatornak?

9. Milyen csomópontok lehetnek egy LabVIEW programban?

10. Hogyan tudunk rekurziót megvalósítani a LabVIEW-ban?

11. Hogyan kapcsoljuk össze az elemeket a Diagram panelon?

12. Honnan tudjuk, hogy az egyes elemek milyen adattípusúak?

13. Hogyan méretezhetjük át elemeinket?

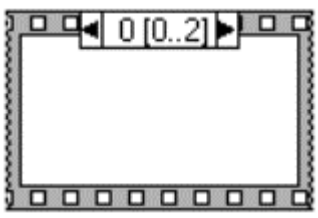
4. Strukturált utasítások

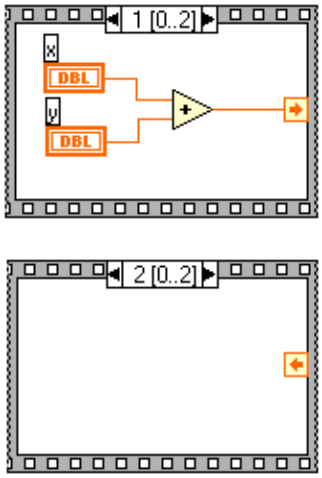
4.1. Szekvencia (Sequence)

A LabVIEW alkalmazásfejlesztő környezetben a program futása nem hasonlít a szöveges programnyelvekben írt programok futására, tehát nem soronként hajtódik végre (mivel nincsenek sorok), hanem párhuzamosan futnak az adatok. Egy adott pillanatban azoknak a csomópontoknak a végrehajtása kezdődik meg, amelyeknek minden bemeneti adata rendelkezésre áll, másrészről egy csomópont mindaddig nem ad ki egyetlen kimenetén sem adatot, ameddig minden számítást el nem végzett. Ha szekvenciát, azaz sorrendi struktúrát alkalmazunk, akkor bizonyos programrészletekhez prioritást rendelhetünk, azaz bizonyos programrészletek előbb hajtódnak végre, mint mások (persze egy szekvencián belül továbbra is párhuzamos az adatforgalom).

A szekvenciát *(4.1. ábra)* a Functions paletta *Structures* tábláján találhatjuk. További szekvencia keretek hozzáadása a struktúrához hasonlóképpen történik, mint a CASE struktúránál, a pop-up menüből kiválaszthatjuk az *Add Frame After* (keret hozzáadása az előző után) vagy az *Add Frame Before* (keret hozzáadása az előző elé) menüpontokat.

Gyakran előfordul, hogy egy előző fázisban számolt értékre a későbbiekben még szükség van. Erre szolgál az *Add Sequence Local* (Szekvencia lokális változó hozzáadás) menüpont. Ha ezt választjuk, megjelenik a szekvencia oldalánál egy kis négyzet, melybe beköthetjük az átadni kívánt adatunkat, ekkor a színe az adattípusnak megfelelően változik, és a képe egy kifele mutató nyíl lesz. A következő kereteken ezt az információt bemenő adatként olvashatjuk le.



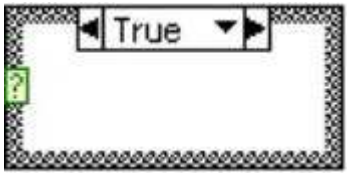


4.1. ábra. Szekvencia

4.2. Elágazás (Case)

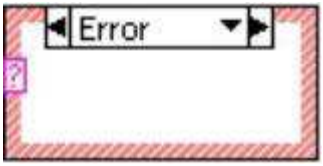
A Case Struktúra a szöveges programnyelvekből ismert feltételek megfelelője. Alkalmazhatjuk kétágú vagy többágú feltételként is.

A Case struktúrát *(4.2. ábra)* a Functions paletta *Structures* tábláján találhatjuk meg. A szelekciónk alapértelmezés szerint kétágú, ekkor egy igaz és egy hamis águnk van. A feltételt, melynek teljesülnie kell ahhoz, hogy az igaz (True) ágban lévő algoritmusrészletünk fusson le, a CASE struktúrát ábrázoló téglalap bal oldalán található kis zöld kérdőjelbe kell kötnünk.



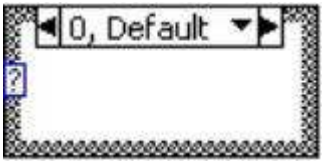
4.2. ábra. Kétágú elágazás

Ha ez a feltétel nem teljesül, azaz az értéke hamis lesz, akkor a hamis (False) ágban lévő programrészletünk hajtódik végre. Hasonlóképpen működik a hiba (Error) vagy nincs hiba (No Error) jel esetén történő végrehajtás *(4.3. ábra)*.



4.3. ábra. Kétágú elágazás hibajelre

Ha többágú szelekcióra *(4.4. ábra)* van szükségünk, akkor a Case struktúra kiválasztása után a keretre bal gombbal kattintunk, és a pop-up menüből kiválasztjuk az *Add Case After* (szelekciós ág hozzáadása az előző után) vagy az *Add Case Before* (szelekciós ág hozzáadása az előző elé) menüpontot. Ekkor a kérdőjel kék színű lesz, tehát a kiválasztás nem logikai, hanem numerikus érték alapján történik.



4.4. ábra. Többágú elágazás

4.3. Ciklusok (Loop)

4.3.1. WHILE ciklus

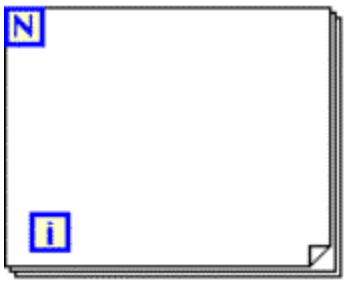
A while ciklus a LabVIEW-ban hasonlóan működik, mint más programnyelvekben, addig ismétli a körülhatárolt programrészletet, míg a futási feltétel értéke igaz.

A feltételes ciklust (While Loop) *(4.5. ábra)* a Functions paletta Structures tábláján találhatjuk. A futási feltételt a ciklust jelképező téglalap jobb alsó sarkában található elembe kell kötnünk. (Az elem a ciklus "területén" belül elmozdítható.) A cikluson belüli programrészletünk addig ismétlődik, míg ez a feltételünk hamis nem lesz. A ciklus elhelyezésekor még egy elemet tartalmaz (), ez az elem tartalmazza azt a számértéket, hogy hányadszor ismétlődött a programrészletünk. A számolás 0-ról kezdődik. Programrészletünket elkészíthetjük a ciklus elhelyezése után, de a már meglévő programrészleteket is "bekeretezhetjük" a ciklust jelképező téglalappal.


4.5. ábra. Feltételes ciklus

4.3.2. FOR ciklus

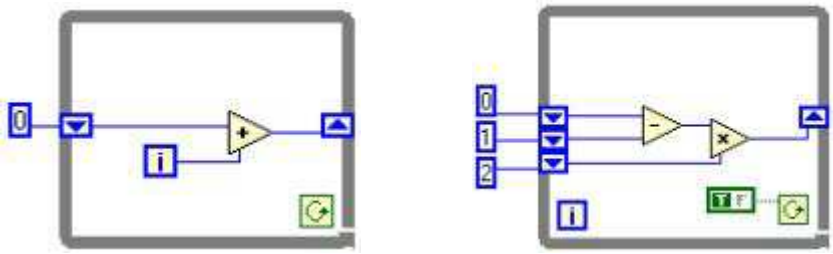
A számlálós ciklus (For Loop) *(4.6. ábra)* adott számszor ismétli a kijelölt programrészletet. A For ciklust a Functions paletta Structures tábláján találhatjuk. Az ismétlési számot a cikluson kívül kell megadnunk, és a struktúra bal felső sarkában található N jelű elembe kell kötnünk. Például azt jelenti, hogy a programrészletünk 15-ször ismétlődjön. A ciklusszámláló 0-ról indul, és a ciklus futásának befejeztével az értéke 14 lesz.


4.6. ábra. Számlálós ciklus

Shift regiszter

A LabVIEW ciklus struktúráinak használata közben gyakran előfordul, hogy egyik ciklusléptésből a következőbe akarunk értéket átadni. Erre szolgálnak a shift regiszterek *(4.7. ábra)*. shift regisztereket úgy hozhatunk létre, hogy a ciklus struktúra bal vagy jobb oldalán kattintva a jobb egérgombbal, a pop-up menüből kiválasztjuk az *Add Shift Register* (shift regiszter hozzáadása) menüpontot. A regiszter kezdőértékét a cikluson kívülről a regiszter bal oldali szimbólumába kötve adhatjuk meg. Kezdőértéket úgy is létrehozhatunk, hogy a shift regiszteren kattintunk az egér jobb gombjával, és a *Create* menüponton belül kiválasztjuk, hogy konstans értéket, kontrollt, vagy indikátort rendelünk a shift regiszterhez.

A ciklus futásakor az az érték, amelyet a jobb oldali shift regiszterhez kötünk, automatikusan átkerül a ciklus következő lefutásakor a shift regiszter bal oldali kapcsolódási pontjára, így itt leolvashatjuk az adatot a következő iterációban a számítások elvégzéséhez. Egy ciklushoz használhatunk több shift regisztert különböző adatokhoz, melyek lehetnek akár különböző típusúak is, de azt is megtehetjük, hogy egy adat korábbi iterációkban volt értékét lekérdezhetjük. Ha a shift regiszteren előhívva a pop-up menüt kiválasztjuk az *Add Element* menüpontot, a ciklus bal oldalán a regiszter újabb része jelenik meg, melyen az előző iteráció előtti értékét olvashatjuk le a bekötött adatunknak. Így több elemet elhelyezhetünk, melyek segítségével egyre "régebbi" adatokat is kiolvashatunk.



(a) (b)

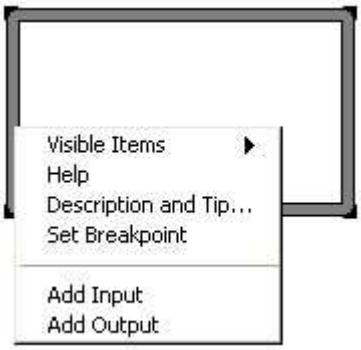
4.7. ábra. A Shift regiszter használata

4.4. Képlet (Formula Node)

A képlet (Formula Node) *(4.8. ábra)* segítségével egy szövegdobozba írhatjuk kiszámítandó aritmetikai képleteinket. Ennek az az előnye, hogy kisebb helyet foglal el egy sokparaméteres képlet, mint a megfelelő grafikus képű műveletek elhelyezése a diagram panelen.

Képletünkhöz szükség van bemenetek és kimenetek definiálására. Ezt a kereten az egér jobb gombjával kattintva, a pup-up menü *Add Input*, ill. *Add Output* menüpontjával tehetjük meg.

A képletben a C nyelv szintaxisának megfelelően adhatjuk meg az összefüggéseket, (például Y=3*X+4/X;) de akár kisebb algoritmusokat is készíthetünk, ha az a számításhoz szükséges.



(a)

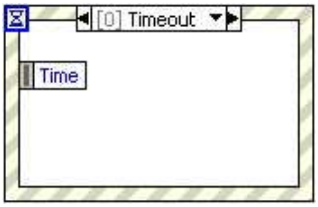
(b)

4.8. ábra. Képlet alkalmazása

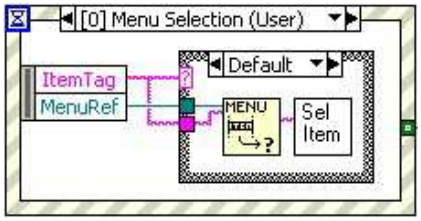
4.5. Esemény (Event)

Az objektumorientált programozás fontos momentuma az események kezelése. Erre szolgál az esemény (Event) struktúra *(4.9. ábra)*, mely a programnak nem egy fix pozíciójában hajtódik végre, hanem akkor, mikor egy adott esemény bekövetkezik (pl. billentyű lenyomás, egérmozgatás stb.).

Az alábbi példán *(4.9. ábra)* egy menüpont kiválasztását kezelő eseményt láthatunk.



(a)



(b)

4.9. ábra. Esemény alkalmazása

4.6. Ellenőrző kérdések

1. Milyen struktúratípusokat használhatunk a LabVIEWban?

2. Mikor használunk szekvenciát?

3. Mit jelent a *Sequence Local* , és mire használjuk?

4. Hogyan változtathatjuk meg az elágazás típusát (kétágú, többágú, stb.)?

5. Mire használhatjuk WHILE ciklus esetén a ciklusváltozót?

6. Tudunk-e végtelen ciklust készíteni?

7. Mire használhatunk végtelen ciklust?

8. Hogyan olvashatjuk ki a ciklus aktuális lépésében egy változó előző ciklusl épésbeli értékét?

9. Honnan kezdődik a FOR ciklus ciklusváltozójának növekedése?

10. Hogyan adhatjuk meg, hogy hányszor fusson le a FOR ciklus?

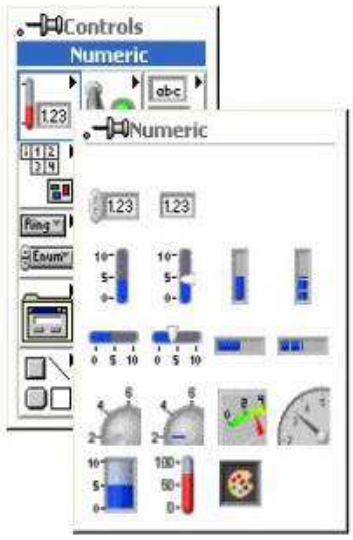
11. Össze tudunk-e állítani minden olyan képletet műveletekből, amelyet a FORMULA NODE-ba beírhatunk?

12. Mit nevezünk eseménynek?

5. Adattípusok és műveletek

5.1. Numerikus elemek

A Controls Palette *(2.14. ábra)* egyik táblája a *Numeric Contols (5.1. ábra).* Itt találhatjuk meg azokat az elemeket, melyek valamilyen szám típusú adattal dolgoznak. Ez, mint láthatjuk is, lehet egyszerű, digitális kijelző, mely használható kontroll és indikátor módban is. Kiválaszthatunk itt konstans értékeket, különböző típusú, vízszintes és függőleges elhelyezkedésű csúszkákat (slide), tartályt, hőmérőt, potmétert, analóg kijelzőket, stb.



5.1. ábra. Numerikus elemek

Minden elemnek saját menüje van *(5.2. ábra)*, melyben a működési és megjelenítési tulajdonságokat állíthatjuk be. A pop-up menü felső része azonos, az elemek közötti eltérés az alsó részben látható.

Az első menüpont a *Visible Items* menüpont, melyben azt állíthatjuk be, hogy legyen-e felirat az elem mellett, legyen-e mértékegység a számérték mögött, látható legyen-e a számrendszer jele vagy sem.

A második menüpont a *Find Terminal*, mely segítségével a Diagram panelen megtalálhatjuk elemünk megfelelő szimbólumát.

A következő a *Change to Indicator*, ha az objektum kontroll módban van, vagy *Change to Control*, ha az elem indikátor módban van, és a működési mód váltására szolgál.

A *Description and Tip* menüpontban rövid leírást készíthetünk elemünkhöz.

A *Create* menüpontban állíthatunk elő lokális változót, és tulajdonság listát, melyekről később lesz szó.

A *Replace* menüpont segítségével kicserélhetjük a nem kívánt elemeinket valamilyen más elemre.

A *Data Operations* menüpontban visszaállíthatjuk értékünket az alapértelmezettre, beállíthatjuk az alapértelmezett értéket, kivághatjuk, másolhatjuk, beilleszthetjük az értékeket, készíthetünk rövid leírást elemeinkhez.

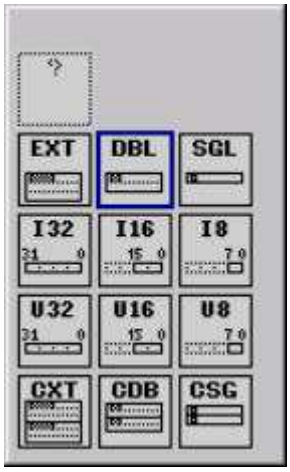
Az *Advanced* menüponttal billentyűkombinációt rendelhetünk az objektumhoz, átszerkeszthetjük, elrejthetjük azt.

5.2. ábra. Elem saját menüje

A *Representation* menüpontban *(5.3. ábra)* választhatjuk ki, hogy numerikus elemünk milyen típusú legyen. Lehet Extended, Double, Single, egész, vagy előjel nélküli típus, melyen 32, 16, vagy 8 számjegyet ábrázolhatunk, ill. komplex típus.

A *Data Range* menüpontban adhatjuk meg az ábrázolni kívánt tartomány minimum és maximum értékét, az alapértelmezett értéket, és megjelenik a kiválasztott típusunk szimbóluma is, illetve meg lehet adni, mi történjen, ha az érték túllépi a megadott tartomány határait.

A *Format & Precision* menüpontban a számábrázolás módját és a kijelzendő karakterek számát adhatjuk meg, illetve azt is beállíthatjuk, hogy az értéket számként vagy dátumként ábrázoljuk.



5.3. ábra. Elem saját menüje

Más elemek esetén, mint arról már szó volt, a pop-up menü felsőrésze azonos funkciókat tartalmaz, viszont az alsó rész kiegészülhet más menüpontokkal is.

A *Representation*, a *Data Range*, és a *Format & Precision* menüpontok itt is hasonló feladatokat látnak el, mint az egyszerű digitális kijelző esetén.

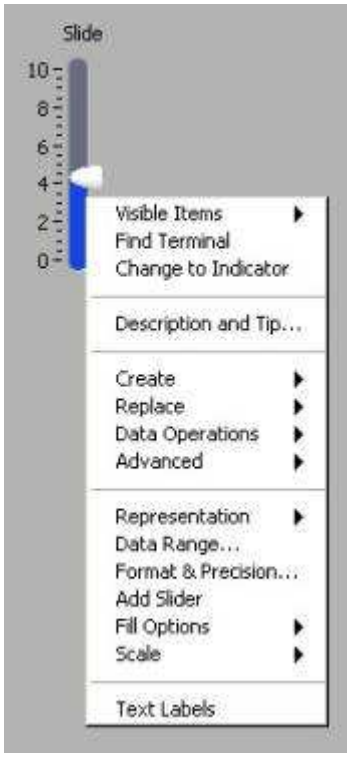
Az *Add Slider* menüpont segítségével összetett elemet állíthatunk elő *(5.4. ábra)* úgy, hogy egy tartományban több csúszka lehet, melyek különböző értékeket vehetnek fel, és mindegyik értékét egy-egy egyszerű digitális kijelzőn követhetjük nyomon.

A *Fill Options* menüpontban azt tudjuk állítani, hogy a tartomány mely része legyen beszínezve: a csúszka alatti, a csúszka feletti, a csúszkák közötti vagy egyik sem.

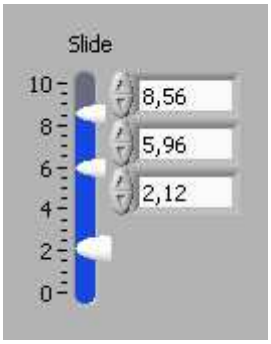
A *Scale* menüpont megtalálható minden elemnél, mely mellett értékskála jeleníthető meg, és itt beállíthatjuk a skála jellemzőit, és számmegjelenítésének tulajdonságait.

A *Text Labels* menüpontot akkor válasszuk, ha a skálán ill. a kijelzőn a számérték helyett szöveges formában kívánjuk megjeleníteni, hogy az aktuális érték a maximumhoz vagy a minimumhoz van-e közelebb.

A skálával rendelkező elemek tartományhatárainak értékét egyszerűen megnövelhetjük, vagy lecsökkenthetjük. Az *Operating Tool* eszközzel a skála valamely szélsőértékére rákattintva átírhatjuk tetszőlegesen az ott szereplő értéket.



(a)

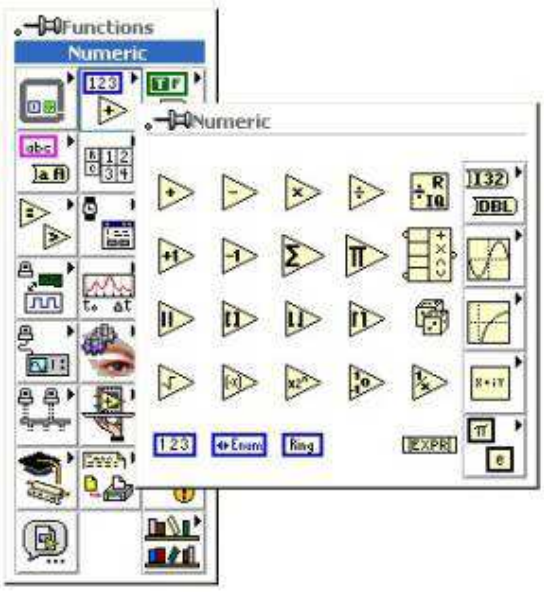


(b)

5.4. ábra. Összetett elem kialakítása

5.2. Műveletek numerikus elemekkel

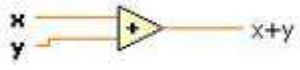
A Functions Palette egyik pontja tartalmazza a numerikus műveleteket *(5.5. ábra)*. Itt megtalálhatjuk az egyszerűbb és bonyolultabb műveleteket egyaránt, találhatunk konvertáló, trigonometrikus és logaritmikus függvényeket, komplex számokkal történő műveleteket, egyszerű és speciális konstansokat.



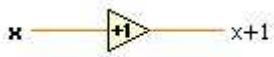
5.5. ábra. Numerikus műveletek

A továbbiakban ezek közül ismerkedünk meg néhány fontosabbal *(5.6. ábra)*.

(a)



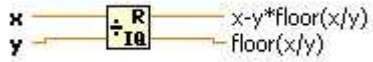
(b)



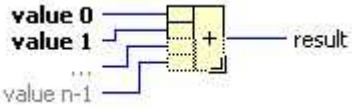
(c)

(d)

(e)



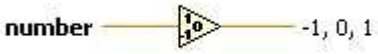
(f)



(g)

(h)

(i)



(j)

(k)

(a) Összeadás (Add)

(b) Inkrementálás (Increment)

(c) Tömb elemeinek összegzése (Add array elements)

(d) Tömb elemeinek szorzata (Multiply array elements)

(e) Osztás maradéka és egészrésze (Quotient and remainder)

(f) Több bemenetű művelet (típusát kiválaszthatjuk) (Compoud arithmetic)

(g) Abszolút érték (Absolute value)

(h) Kerekítés (Round to nearest)

(i) Véletlen szám (Random number)

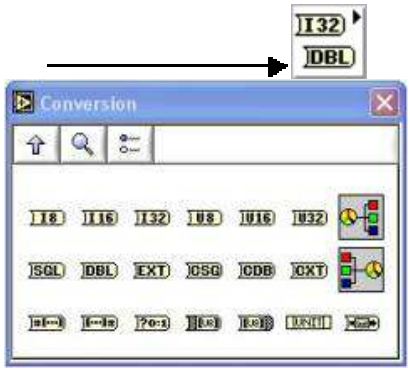
(j) Előjel (Sign)

(k) Konstansok

5.6. ábra. Fontosabb numerikus műveletek

5.2.1. Konverziós műveletek

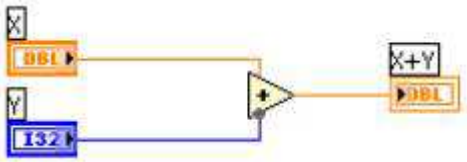
Ha szükségünk van rá, két szám között alkalmazhatunk konverziót *(5.7. ábra)*, egész számból konvertálhatunk lebegőpontossá, vagy fordítva, előjelesből képezhetünk előjel nélkülit. Sőt, string-ből képezhetünk numerikus tömböt, ill. fordítva, de akár logikai tömbből is lehet szám, és fordítva. Logika típusból konvertálhatunk 1, vagy 0 értékké is.



5.7. ábra. Konverziós műveletek

Konverzió történhet automatikusan is *(5.8. ábra)*, ha különböző típusú adatokat kötünk össze. Például összeadunk egy egész és egy lebegőpontos típusú számot:

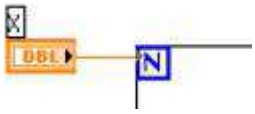
Ebben az esetben a végeredmény a nagyobb pontosságot megengedő típusú lesz.



5.8. ábra. Automatikus konverzió

Ha az eredmény típusa adott, akkor mindenképpen ahhoz kell alkalmazkodni. Ilyen eset lehet, ha a For ciklusnak azt akarjuk megadni, hogy fusson le 4,56-szor *(5.9. ábra)*.

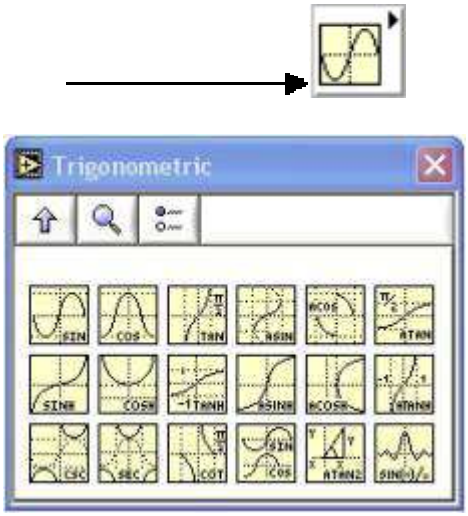
Ebben az esetben kerekítés történik, tehát 4,56-os érték esetén 5-ször fog lefutni a ciklus.



5.9. ábra. Automatikus konverzió For ciklus esetén

5.2.2. Trigonometrikus függvények

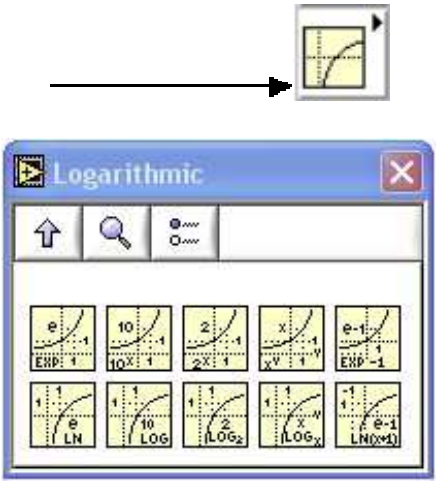
Ebben a csoportban találhatjuk a trigonometrikus függvényeket *(5.10. ábra)*, melyek egy numerikus bemenetből számolják kimeneti értékeiket.



5.10. ábra. Trigonometrikus függvények

5.2.3. Logaritmikus függvények

Ebben a csoportban találjuk a logaritmikus és exponenciális függvényeket *(5.11. ábra)*, ezek is általában egy bemenetű műveletek, kivéve a hatványozást (Power of X) *(5.12. ábra)*.

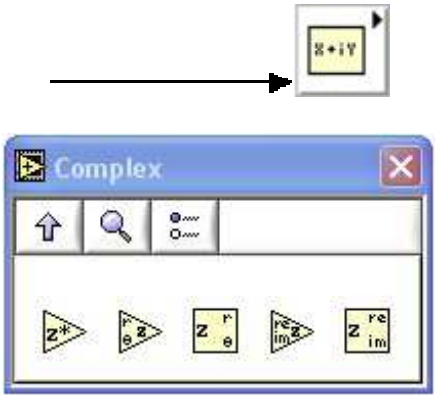


5.11. ábra. Logaritmikus és exponenciális függvények

5.12. ábra. Hatványozás

5.2.4. Komplex számok műveletei

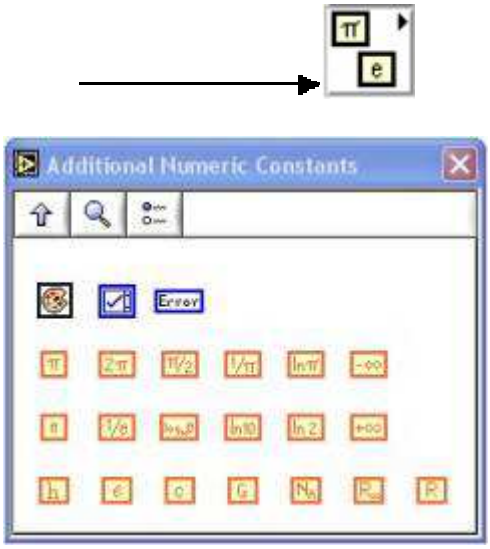
Itt a komplex számok alapvető műveleteit találhatjuk *(5.13. ábra)*, mint például komplex konjugált képzés, valós és képzetes részből komplex képzés, polárból komplex képzés, stb..



5.13. ábra. Komplex számok műveletei

5.2.5. Konstansok

Ebben a csoportban speciális matematikai konstansok közül válogathatunk *(5.14. ábra)*.



5.14. ábra. Matematikai konstansok

5.3. Feladatok

1. Készítsünk programot, mely két bemenő paraméter (az oldalhosszak) értéke alapján kiszámolja egy téglalap területét és kerületét!

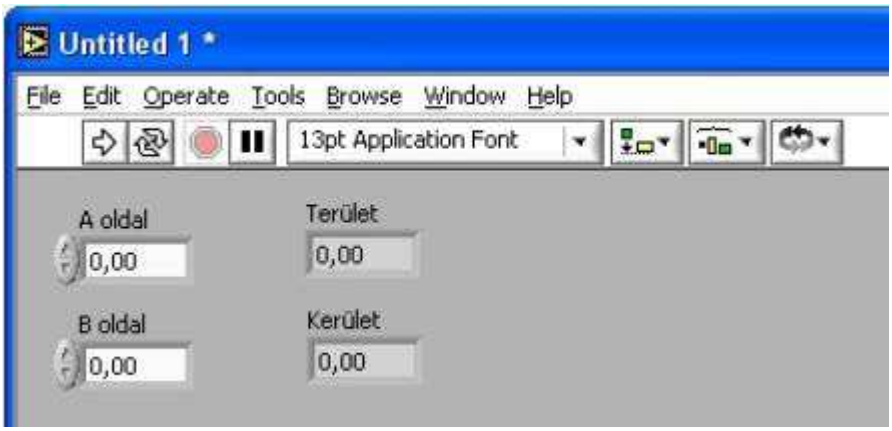
Megoldás:

Mint minden feladatmegoldás elején, helyezzük el a megfelelő vezérlő (Control) és megjelenítő (Indicator) elemeket a felhasználói felületen (Front panel)! Mivel két bemenő adatunk van, és ezek numerikus elemek, ezért a *Controls Palette* numerikus elemei közül kell vezérlőelemeket választanunk. Azt, hogy milyen kinézetű elemet válasszunk (digitális kijelző, tartály, mutatós műszer, stb.), mindig a feladat jellege szabja meg. Javaslom ehhez a feladathoz a digitális kijelzők használatát *(5.15. ábra)*.

Amint egy elemet a Front panelre helyeztünk, rögtön célszerű nevet is adni neki, mert ekkor még nem kell semmilyen eszközt használnunk, később viszont a címke szövegének változtatásához a *Labeling tools* eszközre van szükség.

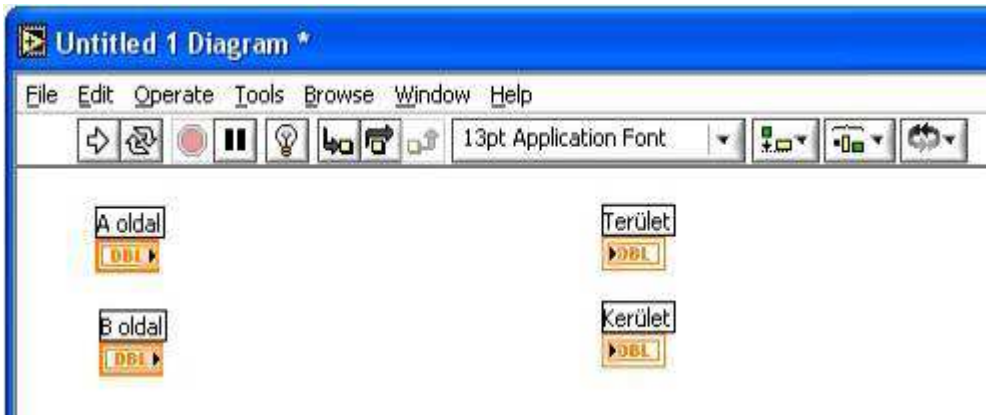
Ha az elhelyezett elemek valamelyike nem vezérlő, hanem megjelenítő elem, akkor ezt a tulajdonságát bármely elemnek a saját pop-up menüjében meg tudjuk változtatni.

Helyezzünk el még két numerikus elemet is, melyek az eredmények megjelenítésére szolgálnak (, tehát indikátorok lesznek).



5.15. ábra. A feladat felhasználói felülete

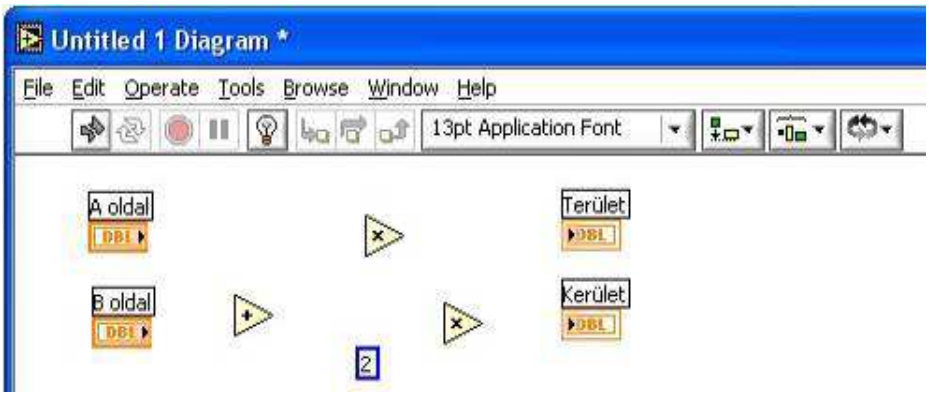
Ha ezzel megvagyunk, a felhasználói felületünket kialakítottuk. Ezután a *Window / Show diagram* menüpontra kattintva (, vagy Ctrl+E) folytassuk a munkát a Diagram panelon *(5.16. ábra)*.



5.16. ábra. Diagram panel az elemek szimbólumaival

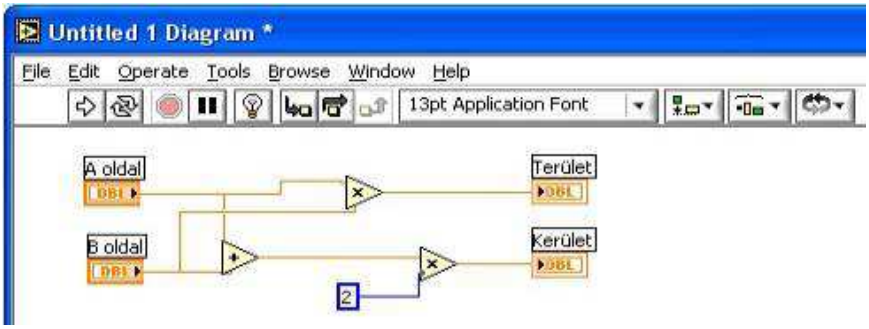
A Diagram panelon megjelennek a Front panelon elhelyezett elemek szimbólumai a megfelelő címkékkel, melyek segítenek az azonosításban. Ez fontos lehet, főképp egy sok elemet tartalmazó program esetén.

A következő lépés a műveletek elhelyezése, és összekötése az elemekkel. Kisebb feladatok esetében megtehetjük, hogy letesszük a panel felületére az összes műveletet, konstanst, amire szükségünk van, így rögtön látjuk, mekkora területen (képernyő-, vagy ablakterület) fér el az algoritmus *(5.17. ábra)*. Nagyobb feladatok esetében ez nem biztos, hogy célravezető, mert a végén már nem emlékszünk, hogy mit miért és hova tettünk. Ekkor kisebb részegységekre bontva, egymás után tesszük meg ezeket a lépéseket.



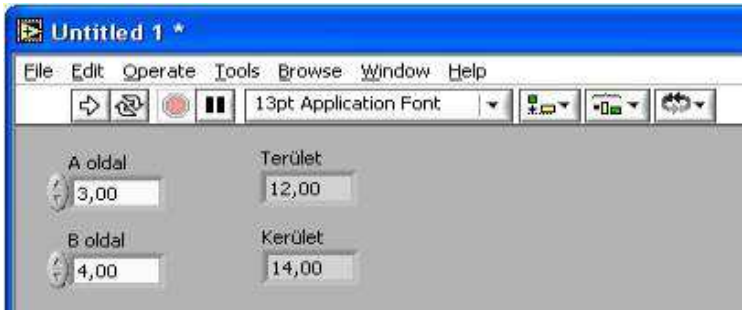
5.17. ábra. Diagram panel az elemek szimbólumaival és a szükséges műveletekkel

Miután elhelyeztük a szükséges műveleteket és konstansokat, össze kell kötnünk őket a *Wiring tool* eszközzel *(5.18. ábra)*. Amíg ezt nem tesszük meg, a programunk nincs futtatható állapotban, ahogyan azt a fenti ábra futtatás gombjának képe is mutatja.



5.18. ábra. Diagram panel az elkészített programkóddal

Ha az elemek összekötését megvalósítottuk, lépjünk vissza a felhasználói felületre, és állítsuk be a bemeneteket a kívánt értékekre az *Operating tool* segítségével, majd futtassuk a programot. A program egyszer lefut, és a megjelenítő elemekről leolvashatjuk a végeredményt *(5.19. ábra)*.

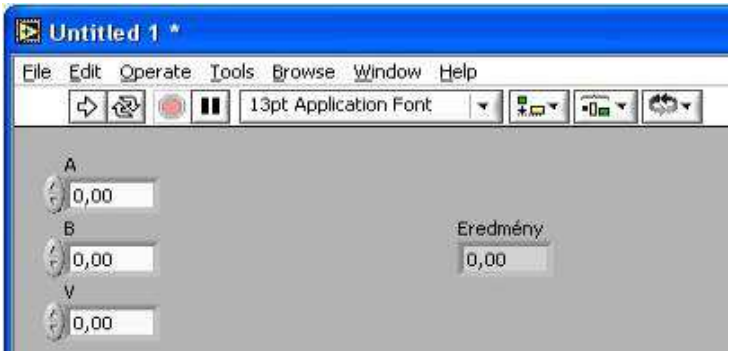


5.19. ábra. Front panel a beállított értékekkel

2. Készítsünk programot, mely két bemenő paraméterrel számítást végez egy harmadik paraméter értéke alapján! A, B bemenő paraméterek, V (választás) szintén bemenő paraméter. Ha V értéke 0, akkor az eredmény A+B legyen, ha V=1, az eredmény A-B legyen, ha V=2, az eredmény A*B legyen, és V=3 értéke esetén A/B-t számoljuk ki!

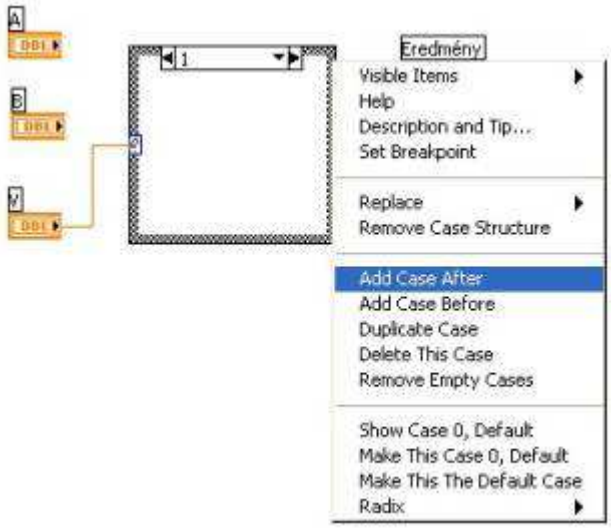
Megoldás:

Helyezzük el a képernyőre A, B, V numerikus vezérlőelemeket, és az Eredmény nevű numerikus megjelenítő elemet *(5.20. ábra)*.



5.20. ábra. Front panel az elhelyezett elemekkel

Ezután átlépünk a Diagram panelre. A megvalósítandó feladat az, hogy V értéke alapján egy többágú szelekciót hozzunk létre. Ehhez egy Case struktúrát kell elhelyeznünk, és a feltétel részbe a V értékét kell kötnünk *(5.21. ábra)*.



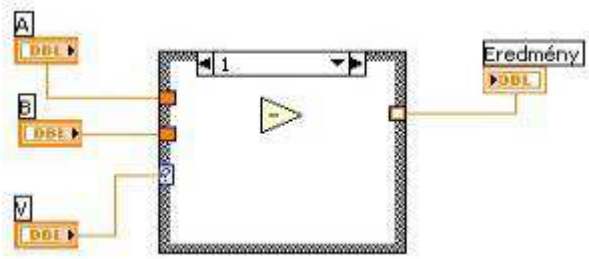
5.21. ábra. Szelekció alkalmazása

Alapértelmezés szerint a Case struktúrának két lapja van, de nekünk négyre van helyezzük. Itt már nem az elemekből, hanem a keret megfelelő csatlakozópontjaiból végezzük el a művelet huzalozását *(5.22. ábra)*.szükségünk. A struktúra keretén kattintva az egér jobb gombjával a pop-up menü *Add Case After* vagy *Add Case Before* menüpontjával még két lappal bővítjük a struktúrát.

A lapok fejlécében lévő számérték jelzi, hogy a feltétel mely értékéhez tartozó lap látható aktuálisan.

Álljunk a 0-s lapra, V=0 esetén A+B a kiszámítandó, tehát ide az összeadás műveletét kell helyeznünk. A művelet bemeneteire kell kötnünk A-t és B-t úgy, mintha a struktúra ott se

lenne. Ugyanígy az Eredményhez kell kötni a művelet kimenetét. Ekkor a struktúra keretén csatlakozópontok keletkeznek. Átváltunk a struktúra 1-es lapjára, ide a kivonás műveletét



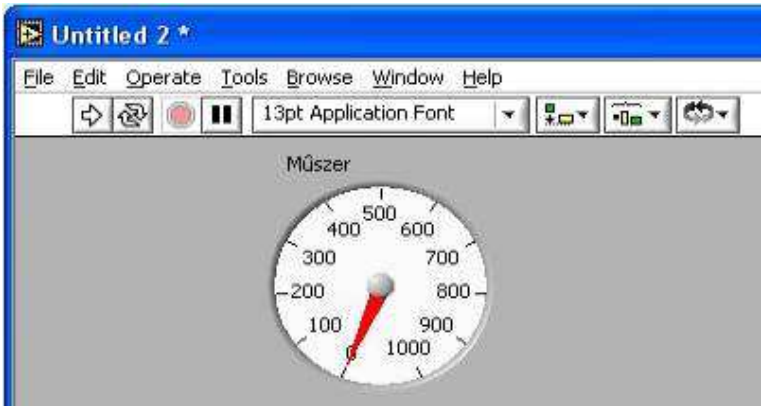
5.22. ábra. Szelekció csatlakozópontjai

Hasonlóképpen alakítjuk ki a 2-es és 3-as lapokat a megfelelő műveletekkel. Ha ez megtörtént, lépjünk át a Front panelre, állítsuk be a bemenetek kívánt értékeit, és futtassuk a programot.

3. Készítsünk programot, melyben egy mutatós műszer mutatóját kilendítjük 0-tól 1000-ig!

Megoldás:

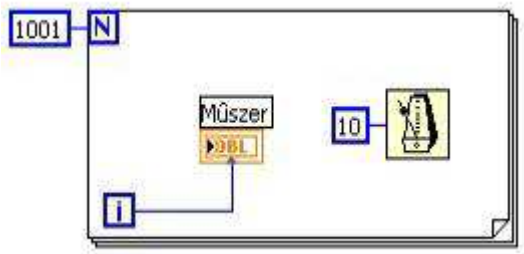
Helyezzünk el a numerikus elemek közül egy mutatós műszert a Front panelen. Ezek az elemek alapértelmezés szerint megjelenítésre szolgálnak, nekünk is erre van szükségünk, de ha a feladat úgy kívánja, az elem saját menüjében vezérlőelemmé változtathatjuk. Elemünk ábrázolási tartománya 0-10 közötti értékek megjelenítésére van beállítva, mivel nekünk 0-tól 1000-ig kell értékeket ábrázolnunk, ezt a tartományt módosítanunk kell. A *Labeling* vagy *Operating tool*-lal kattintsunk a műszer végértékére, és írjuk át azt 1000-re, majd ENTER-rel zárjuk le a módosítást *(5.23. ábra)*.



5.23. ábra. Front panel egy mutatós műszerrel

Lépjünk át a Diagram panelre, és For ciklussal keretezzük be az elemünk szimbólumát. A For ciklusnak 1001-szer kell lefutnia (0-1000), ezért a ciklus bal felső sarkában található N jelű bemenetre kell kötnünk egy 1001 értékű konstanst. Ezt úgy is megtehetjük, hogy az N jelű bemeneten kattintunk az egér bal gombjával, és a *Create Constant* menüpontot választjuk. Ezután a ciklusváltozót rákötjük a Műszer szimbólumára. Mivel számítógépünk túl gyors

ahhoz, hogy követni tudjuk ennek az 1001 műveletnek az eredményét, és csak a végeredményt fogjuk látni, célszerű időkésleltetést tenni programunkba. Ezt a *Functions palette Time & Dialog* csoportjában találjuk. Erre a feladatra a *Wait Until Next ms Multiple* művelet a legalkalmasabb, melynek bemenő paramétere a várakozás ideje milliszekundumban *(5.24. ábra)*.



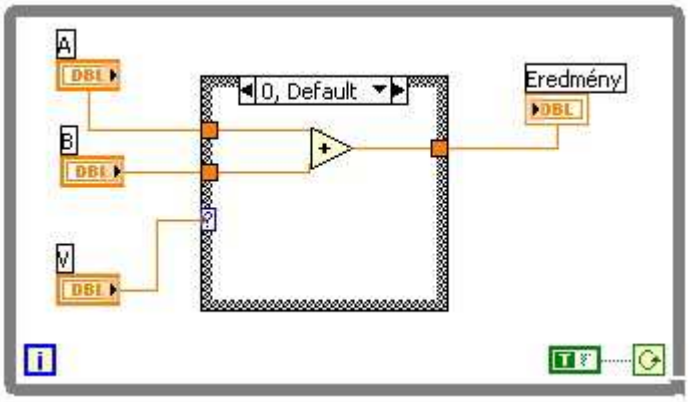
5.24. ábra. A program kódja a diagram panelen

Ha ezzel készen vagyunk, lépjünk át a Front panelre, és futtassuk a programot.

4. Készítsük el a 2-dik feladat olyan verzióját, melyben a V értékét a program futása közben is változtathatjuk, egészen addig, míg le nem állítjuk a programot az ikonsorban található STOP gomb segítségével!

Megoldás:

Töltsük be a 2-dik feladatunkra készített programot, és lépjünk a Diagram panelra. Keretezzük be programunkat egy While ciklussal, és a ciklusfeltételbe kössünk egy igaz konstanst. Így végtelen ciklust készítettünk, melyből csak az ikonsor STOP gombjának lenyomásával tudunk kilépni *(5.25. ábra)*.



5.25. ábra. A program kódja a diagram panelen

Lépjünk át a Front panelra, indítsuk el a programot, és a bemenő adatokat tetszés szerint változtatva az eredményünk automatikusan módosul.

5. Készítsünk programot, mely tetszőleges N-ig (, ahol N a felhasználó által megadott egész szám) kiszámolja a természetes számok összegét!

Megoldás:

Helyezzünk el a Front panelen egy numerikus digitális vezérlő elemet, ez lesz N, és egy numerikus digitális megjelenítő elemet, ez az eredmény lesz.

Lépjünk át a Diagram panelra. A feladat megoldásához For ciklusra lesz szükségünk, melynek a bemeneti paramétere az N kontrol lesz, tehát neki a cikluson kívül kell lennie. A cikluson belül végezzük a számításokat, majd utána kiíratjuk a végeredményt, tehát az eredménynek is a cikluson kívül kell lennie.

A cikluson belül a ciklusváltozó értéke 0-tól N-1-ig egyesével nő. Ha eggyel növeljük, akkor 1-től N-ig kapunk egész számokat. Ha őket sorban összeadjuk, már meg is oldottuk a feladatot. Hagyományos programnyelveknél ezt a feladatot a következőképp oldanánk meg:

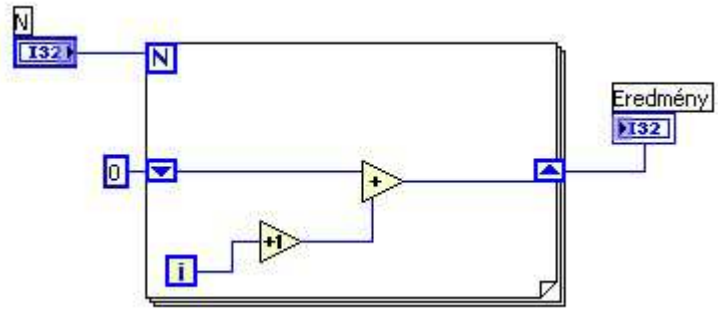
Összeg:=0

Ciklus i:=0-tól N-1-ig

    Összeg:=Összeg+(i+1)

Ciklus vége

LabVIEW-ban az ilyen jellegű feladatoknál a ciklus keretének oldalán elhelyezett *Shit regisztert* használunk. Ezt a keret oldalán kattintva az egerünk jobb gombjával, a pop-up menü *Add Shift Register* menüpontjának kiválasztásával tudjuk a ciklushoz rendelni. A regiszternek a keret bal oldalán megjelenő képe jelenti a változó előző ciklusléptésbeli értékét, a jobb oldali része az új értéket. A cikluson kívül a regiszter bal oldali szimbólumánál kell megadnunk a kezdőértéket, és jobb oldali képéből olvashatjuk ki a végeredményt *(5.26. ábra)*.



5.26. ábra. A program kódja a diagram panelen

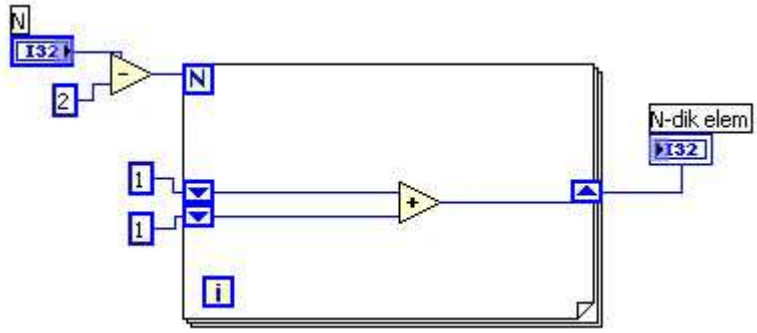
Ha a ciklusban elhelyeztük a szükséges műveleteket és összekötöttük őket a megfelelő bemenetekkel, ill. kimenetekkel, lépjünk át a Front panelre, állítsuk be a bemenet kívánt értékét, és futtassuk a programot.

6. Készítsünk programot a Fibonacci sorozat tetszőleges N-dik elemének meghatározására!

Megoldás:

Mint tudjuk, a Fibonacci sorozat első két eleme 1, a további elemek értéke az előző kettő összege.

Ha a ciklusban nem csak az előző ciklusléptésbeli értéket szeretnénk tudni, hanem az azelőttit is, akkor a Shift regiszternek egy újabb elemét kell feltüntetnünk. Ezt a regiszter pop-up menüjében, az *Add Element* menüpont kiválasztásával tehetjük meg. (Íly módon, a regiszter több elemének hozzáadásával akár még korábbi értékeket is kiolvashatunk.) A regiszter mindkét baloldali képének adjunk kezdőértéket, és mivel az első két elemet ezzel meg is határoztuk, a ciklusnak már csak N-2-szer kell lefutnia. A cikluson belül adjuk össze a regiszter két baloldali képének értékeit, és kössük az eredményt a jobboldali képbe *(5.27. ábra)*.



5.27. ábra. A program kódja a diagram panelen

Ha elvégeztük az összekötéseket, lépjünk át a Front panelre, állítsuk be a bemenet kívánt értékét, és futtassuk a programot.

5.4. Logikai elemek

A logikai elemeket a *Controls* paletta *Boolean* tábláján találhatjuk *(5.28. ábra)*, melyek lehetnek nyomógombok, kapcsolók, LED-ek, rádiógombok, check boxok, és egyéb olyan elemek, melyek kétféle értéket vehetnek fel. A logikai elem értéke lehet igaz (True), vagy hamis (False).

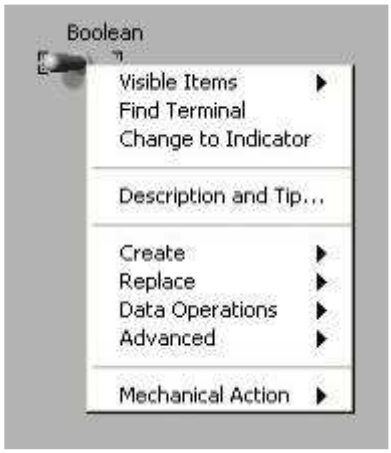
5.28. ábra. Logikai elemek

Bármelyik elemet elhelyezve a felhasználói felületen, beállíthatjuk annak egyedi tulajdonságait. A logikai elemek pop-up menüje nagyon hasonlít a numerikus elemek menüjére *(5.29. ábra).* A menüpontok nagy része megegyezik a numerikus elemeknél látott beállítási lehetőségekkel

Az első menüpont a *Visible Items* menüpont, melyben azt állíthatjuk be, hogy legyen-e felirat az elem mellett, legyen-e mértékegység a számérték mögött, látható legyen-e a számrendszer jele vagy sem.

A második menüpont a *Find Terminal*, mely segítségével a Diagram panelen megtalálhatjuk elemünk megfelelő szimbólumát.

A következő a *Change to Indicator*, ha az objektum kontroll módban van, vagy *Change to Control*, ha az elem indikátor módban van, és a működési mód váltására szolgál.



5.29. ábra. Logikai elemek saját menüje

A *Description and Tip* menüpontban rövid leírást készíthetünk elemünkhöz.

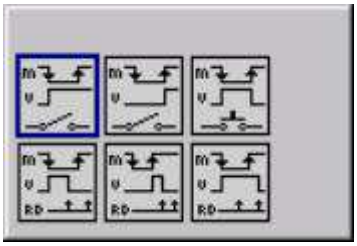
A *Create* menüpontban állíthatunk elő lokális változót, és tulajdonság listát, melyekről később lesz szó.

A *Replace* menüpont segítségével kicserélhetjük a nem kívánt elemeinket valamilyen más elemre.

A *Data Operations* menüpontban visszaállíthatjuk értékünket az alapértelmezettre, beállíthatjuk az alapértelmezett értéket, kivághatjuk, másolhatjuk, beilleszthetjük az értékeket, készíthetünk rövid leírást elemeinkhez.

Az *Advanced* menüponttal billentyűkombinációt rendelhetünk az objektumhoz, átszerkeszthetjük, elrejthetjük azt.

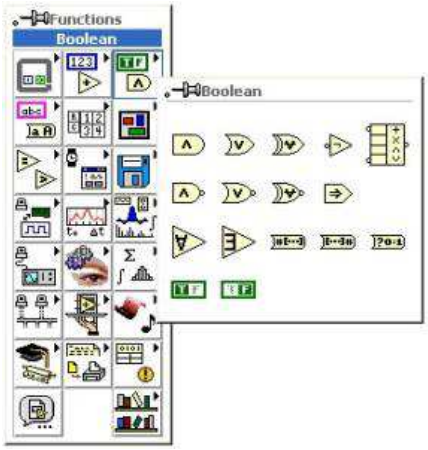
A *Mechanical Action* menüpont csak a logikai elemekre jellemző menüpont, mely a logikai érték váltásának időpillanatát, ill. a váltás hosszát határozza meg *(5.30. ábra).* A programkészítő a feladat jellegétől függően választhatja ki a legmegfelelőbb működési tulajdonságot.



5.30. ábra. Működési jellemzők

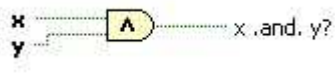
5.5. Műveletek logikai elemekkel

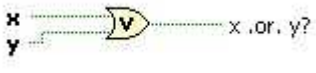
A logikai elemekkel végezhető műveletek a Functions paletta *Boolean* tábláján találhatók *(5.31. ábra)*. Itt helyezkednek el az alapvető logikai műveletek, konverziók, összetett logikai műveletek.

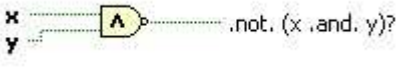


5.31. ábra. Logikai műveletek

A *5.32. ábrán* ezekről a műveletekről láthatunk rövid összefoglalást.

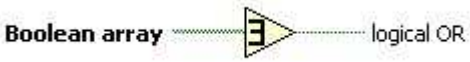
(a) 

(b) 

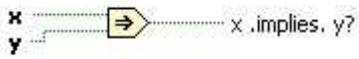
(c) 

(d) 

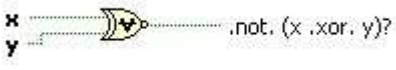
(e) 

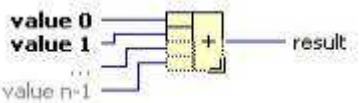
(f) 

(g) 

(h) 

(i) 

(j) 



(k)

(l)

(m)

(a) ÉS művelet két elem között

(b) VAGY művelet két elem között

(c) ÉS NEM művelet két elem között

(d) VAGY NEM művelet két elem között

(e) ÉS művelet egy tömb elemei között

(f) VAGY művelet egy tömb elemei között

(g) Elem negálása

(h) Implementáció

(i) KIZÁRÓ VAGY művelet

(j) KIZÁRÓ VAGY művelet negáltja

(k) Több bemenetű művelet

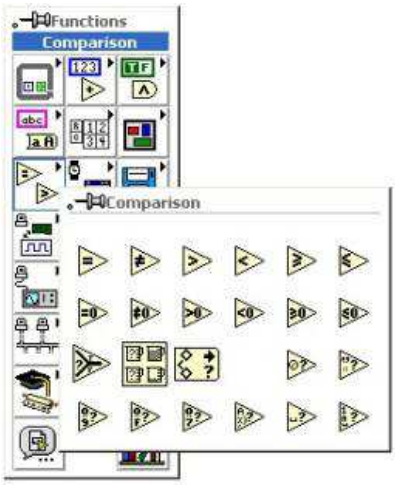
(l) Konverziós műveletek

(m) Logikai konstansok

5.32. ábra. Alapvető logikai műveletek

5.6. Összehasonlító műveletek

Az összehasonlító műveleteket a Functions paletta *Comparison* tábláján találhatjuk *(5.33. ábra)*. Ezek a műveletek többnyire numerikus értékek valamiféle szempont szerinti összehasonlítása alapján logikai kimenetet állítanak elő.



5.33. ábra. Összehasonlító műveletek

Az első sorban két bemeneti paraméterű műveleteket láthatunk, melyek jelentése mindenki számára egyértelmű lehet: egyenlő, nem egyenlő, kisebb, nagyobb, kisebb egyenlő, nagyobb egyenlő.

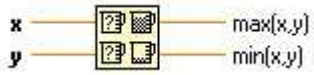
A második sorban egyparaméteres műveleteket láthatunk, mely az előző műveleteket alkalmazza a bemenő adat értéke és a 0 közötti összehasonlításra.

A harmadik sor első három eleme igen hasznos, és gyakran lehet rájuk szükségünk a programkészítés során, így ezekkel részletesebben is foglalkozunk. A többi elem a bemenetet vizsgálja különböző szempontok szerint: a bemenet decimális szám-e, a bemenet hexadecimális szám-e, a bemenet oktális szám-e, a bemenet nyomtatható ASCII karakter-e, a bemenő karakter szóköz-e, stb..

SELECT művelet (5.34. ábra): működése hasonlít a kétágú szelekció működéséhez, ha az S feltétel igaz, akkor a kimeneten a T érték olvasható le, ha viszont S hamis, akkor a kimeneten az F érték jelenik meg.

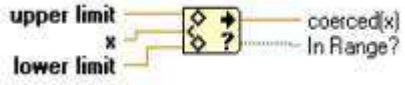
5.34. ábra. SELECT művelet

MIN & MAX művelet (5.35. ábra): a bemenetre kötött X és Y érték közül a nagyobbik érték a jobb felső kimeneten, a kisebbik a jobb alsó kimeneten olvasható le.



5.35. ábra. MIN & MAX művelet

IN RANGE művelet (5.36. ábra): megvizsgálja, hogy X a két szélsőérték által meghatározott tartományban van-e. Coerced(X) értéke X, ha X a tartományon belül van, ill. upper limit vagy lower limit, ha X a tartományon kívül van.



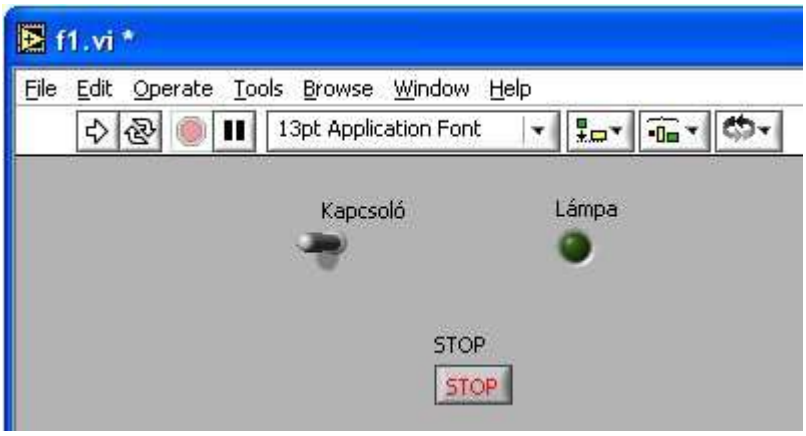
5.36. ábra. IN RANGE művelet

5.7. Feladatok

1. Készítsünk programot, melyben egy kapcsolót fel-le kapcsolgatva egy lámpa világít, vagy nem világít egészen addig, míg a felhasználói felületen elhelyezett STOP gombot meg nem nyomjuk!

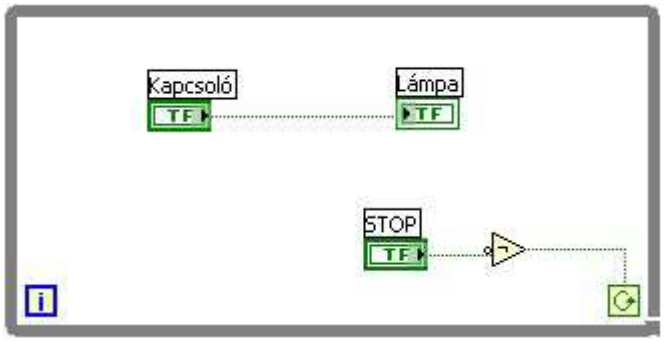
Megoldás:

Helyezzük el a felhasználói felületen a kapcsolót, a lámpát, és a STOP gombot. Ezek közül a kapcsoló és a gomb kontrol legyen, a lámpa pedig indikátor *(5.37. ábra)*.



5.37. ábra. A program felhasználói felülete

A Diagram panelen kössük össze a kapcsolót a lámpával. Ezután már csak a folyamatos működésről kell gondoskodnunk. Kerítsük be az eddigieket egy WHILE ciklussal, és a ciklusfeltételbe kössük be a STOP gomb negáltját, így addig fut a ciklus, míg meg nem nyomjuk a STOP gombot *(5.38. ábra)*.



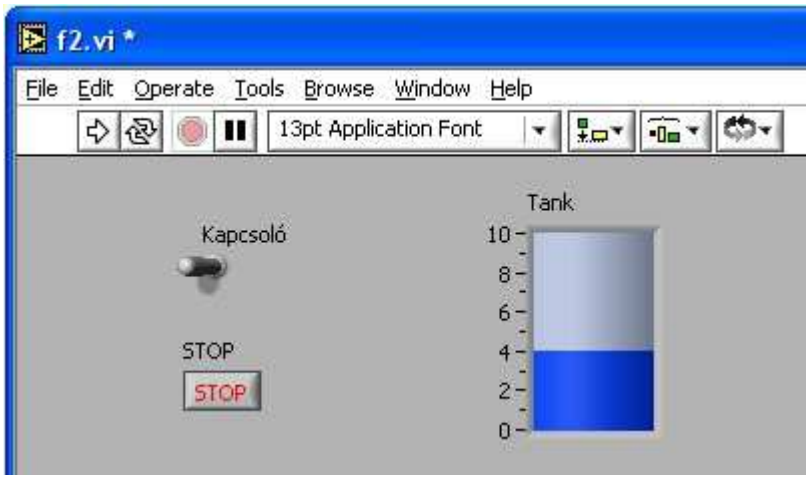
5.38. ábra. A program forráskódja a Diagram panelen

Átlépve a Front panelre teszteljük programunkat.

2. Készítsünk programot, melyben egy tartály folyadékszintjét növeljük, vagy csökkentjük egy kapcsoló bekapcsolt, ill. kikapcsolt állapota alapján! A program addig fusson, míg a felhasználói felületen elhelyezett STOP gombot meg nem nyomjuk!

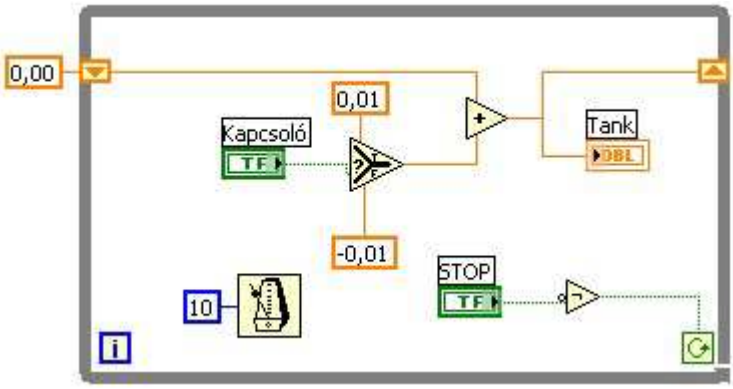
Megoldás:

Helyezzük el a felhasználói felületen a kapcsolót, a tartályt, és a STOP gombot. Ezek közül a kapcsoló és a gomb kontrol legyen, a tartály pedig indikátor *(5.39. ábra)*.



5.39. ábra. A program felhasználói felülete

A Diagram panelen helyezzünk el egy WHILE ciklust, és kössük a ciklusfeltételbe a STOP gomb negáltját. A kapcsoló igaz vagy hamis állásától függően pozitív, vagy negatív számmal növeljük a tartájunk előző értékét. Ezt megtehetjük elágazással, de használhatjuk az összehasonlító műveletek közül a select utasítást is, ami ebben az esetben egyszerűbb megoldásnak tűnik. A tartály értékének folyamatos módosítása shift regiszterrel oldható meg. Hogy követni tudjuk szemünkkel az eseményeket, rakjunk programunkba időkésleltetést is *(5.40. ábra)*.



5.40. ábra. A program forráskódja a Diagram panelen

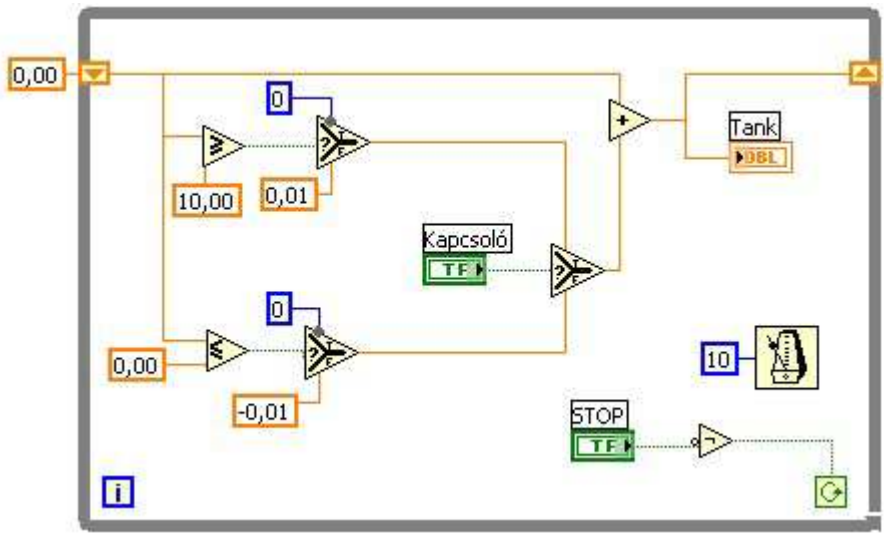
Miután kialakítottuk programunk forráskódját, lépjünk át a Front panelre és teszteljük annak működését.

3. A 2-es feladathoz elkészített programunkat finomítsuk úgy, hogy ha megtelt a tartály, ne töltődjön tovább, és ha leürült, ne ürüljön tovább!

Megoldás:

A tartályunknak van egy alsó és egy felső értékhatára. Az előző feladat megoldása szerint, ha bekapcsolva felejtjük kapcsolónkat, akkor a tartály értéke túllépheti a határértékét. Ezt úgy akadályozhatjuk meg, hogy figyeljük a tartály aktuális értékét, és ettől függően döntjük el,

hogy bekapcsolt kapcsolóállás esetén kell-e tölteni vagy sem, ill. kikapcsolt kapcsolóállás esetén kell-e üríteni vagy sem *(5.41. ábra)*.



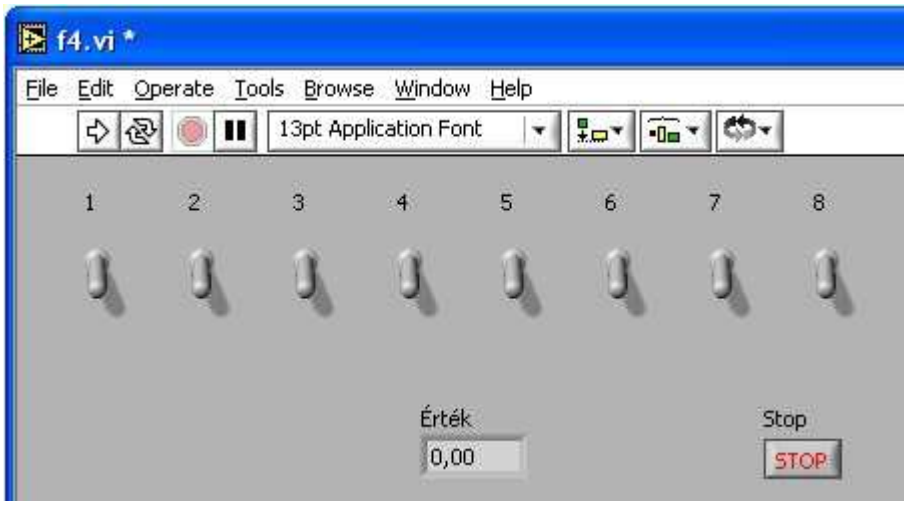
5.41. ábra. A program forráskódja a Diagram panelen

A felhasználói felületen elindítva programunkat vizsgáljuk meg, mi lesz a működésbeli különbség a mostani, és az előző feladat megoldása között.

4. Készítsünk olyan programot, melynek felhasználói felületén elhelyezünk 8 kapcsolót, és egy digitális kijelzőt, és a digitális kijelzőn megjelenítjük a bekapcsolt állapotú kapcsolók darabszámát!

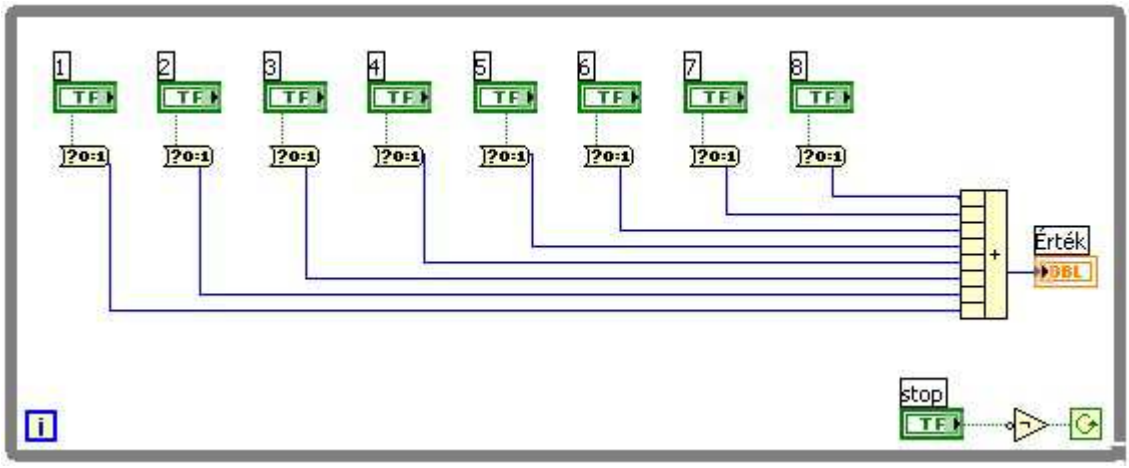
Megoldás:

Legyen a programunk olyan működésű, hogy folyamatosan változtathassuk bemeneteinket. Ehhez egy STOP gomb és egy WHILE ciklus szükséges. Helyezzük el a felhasználói felületre a 8 kapcsolót és a kijelzőt, ill. a STOP gombot *(5.42. ábra)*.



5.42. ábra. A program felhasználói felülete

A Diagram panelen helyezzünk el *Boolean to (0, 1)* műveletet, mely a logikai értéket numerikussá konvertálja. Ezt másoljuk le még 7 példányban, kössük össze a kapcsolók szimbólumaival, és a kimeneteiket adjuk össze a *Compoud arithmetic* művelet segítségével *(5.43. ábra)*.



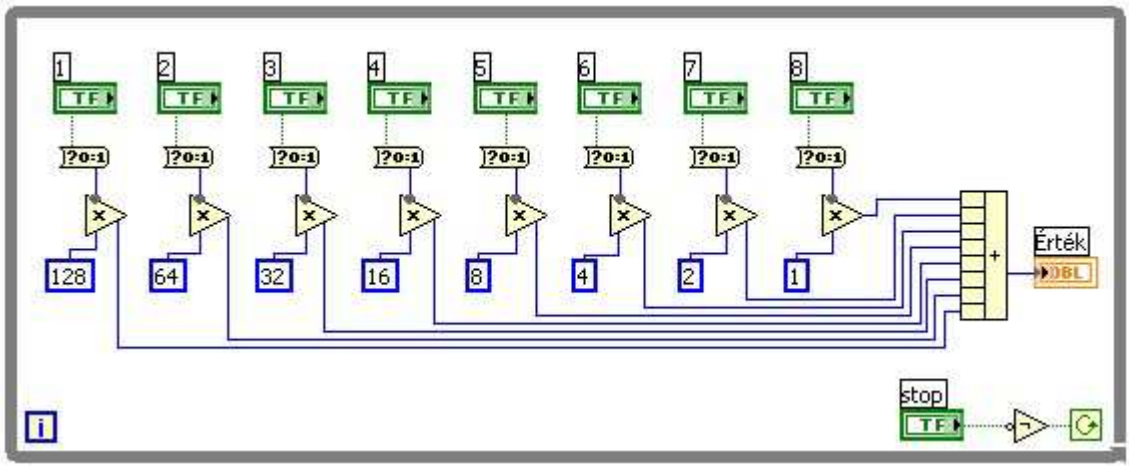
5.43. ábra. A program forráskódja a Diagram panelen

Miután kialakítottuk programunk forráskódját, lépjünk át a Front panelre és teszteljük annak működését.

5. Alakítsuk át az előző programot úgy, hogy a kapcsolók száma helyett a kapcsolókkal kifejezett bináris szám decimális értéke jelenjen meg a kijelzőn!

Megoldás:

Mielőtt a konverziós műveletek kimeneteit összegeznénk, szorozzuk meg őket a bináris számrendszer helyi értékeinek megfelelő értékkel *(5.44. ábra)*.



5.44. ábra. A program forráskódja a Diagram panelen

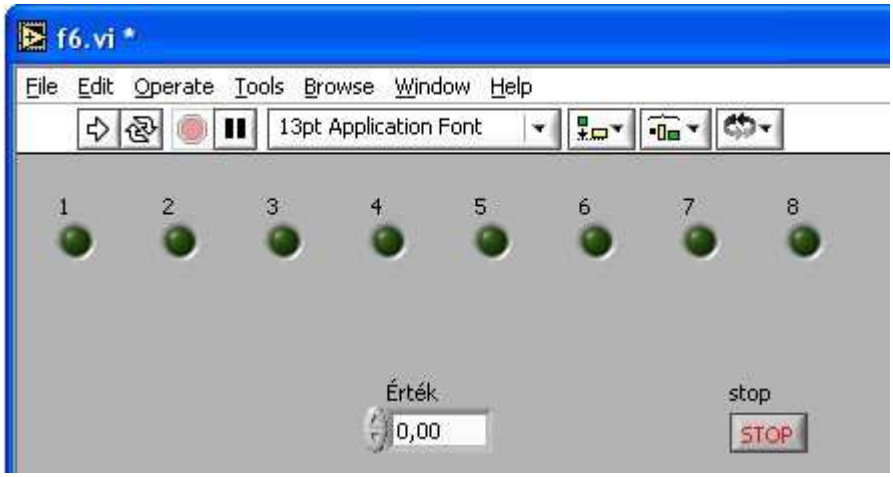
Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

6. Készítsünk programot, mely a megadott decimális számot megjeleníti bináris alakban lámpák segítségével! A beolvasott érték ne legyen nagyobb 255-nél, így 8 lámpával megoldható a feladat.

Megoldás:

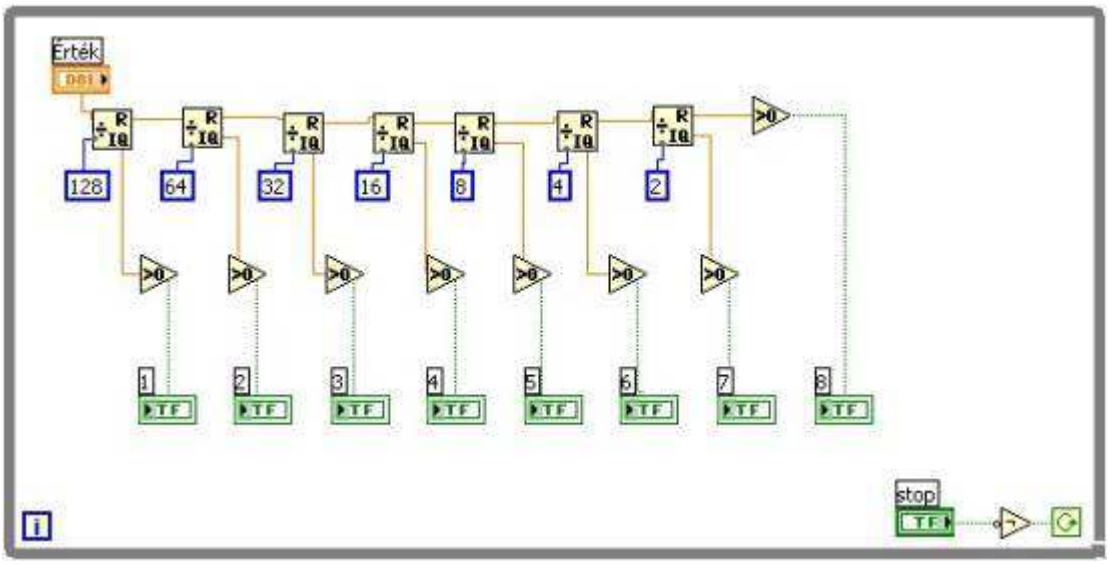
A feladat megoldása könnyebb lenne, ha a lámpákat logikai tömbként kezelhetnénk, de a tömbök használata csak később következik, így más megoldást kell találnunk.

Helyezzük el a numerikus vezérlőelemünket, egy STOP gombot, és a lámpákat a felhasználói felületen *(5.45. ábra)*, majd lépjünk át a Diagram panelre.



5.45. ábra. A program felhasználói felülete

Az előző feladatban használt konverziós művelet logikai értékből állított elő 0-át vagy 1-et. Ennek a fordítottját nem találjuk meg, viszont azt mondhatjuk, ha a szám értéke nagyobb mint 0, az legyen igaz, egyébként hamis. A számunkat a legnagyobb helyi értéktől lefelé végigosztjuk a helyi érték számával úgy, hogy az osztások eredményeként keletkezett maradékot osztjuk tovább, akkor 1-eseket vagy 0-kat kapunk eredményül, amiket az előbb említett módon logikai értékké konvertálunk *(5.46. ábra)*. Mivel az első osztást 128-al fogjuk végezni, akkor kapunk helyes eredményt, ha a beolvasott szám nem nagyobb 255-nél.



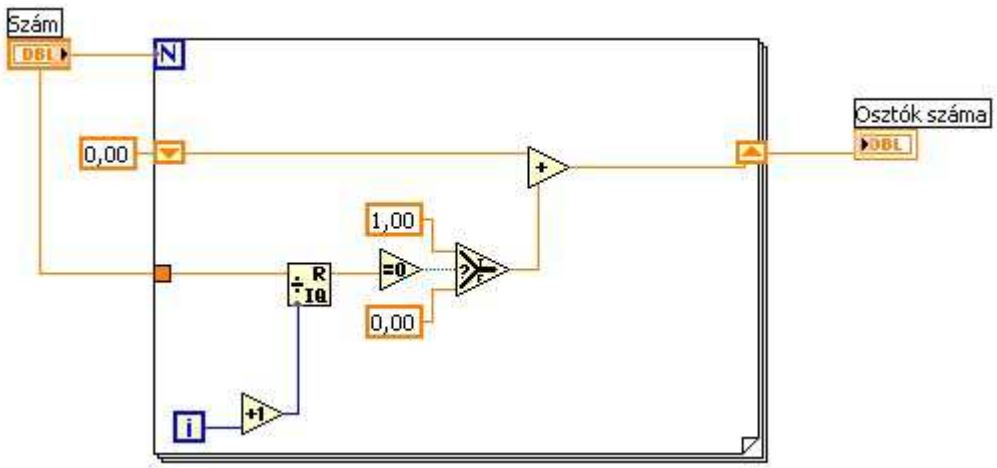
5.46. ábra. A program forráskódja a Diagram panelen

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

7. Készítsünk programot, mely meghatározza a felhasználó által megadott decimális szám osztóinak számát!

Megoldás:

A legegyszerűbb megoldás, ha 1-től a számig (, mely pozitív egész szám legyen) elosztjuk a beolvasott számot a ciklusváltozóval, és megvizsgáljuk a maradékot. Ha a maradék 0, akkor a szám osztható az adott értékkel, és növeljük az osztók számát egyel. Az összegzést shift regiszterrel oldjuk meg, és a legelején az összeget nullázzuk. A végeredményt a ciklus lefutása után a regiszter jobb oldali eleméből olvashatjuk ki *(5.47. ábra).*

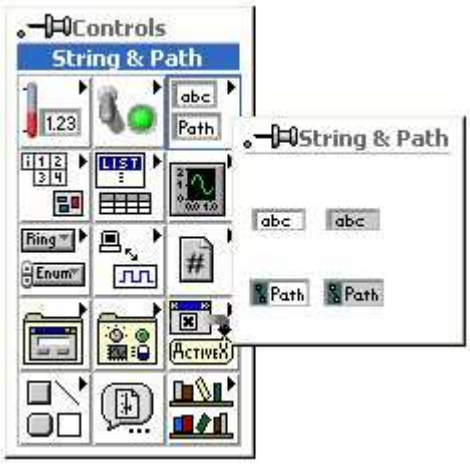


5.47. ábra. A program forráskódja a Diagram panelen

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

5.8. Szöveg típusú elemek

A szöveg típusú adat bemeneti és kimeneti elemek a Controls paletta *String & Path* tábláján találhatók *(5.48. ábra)*. Egy szöveg típusú adat megjeleníthető és nem megjeleníthető karakterek halmazából állhat. Szöveg típusú adatokat használunk a fájl műveleteknél, a parancs-, ill. adatbeviteli műveleteknél és a hálózati kapcsolatban, és üzenetek megjelenítésénél.



5.48. ábra. Szöveg típusú elemek

Szöveges típusú elem méretét tetszőlegesen állíthatjuk a *Positioning Tool* eszköz segítségével.

Egy szöveges típus esetén a pop-up menü *(5.49. ábra)* felső része hasonló funkciókat lát el, mint a már megismert típusoknál.

A *Visible Items* menü kiegészült a *Scrollbar* menüponttal, mely akkor használható, ha a szöveges elem függőleges mérete elég nagy ahhoz, hogy megjeleníthessünk egy az elemhez tartozó gördítősávot.

A szöveges adatunk alapértelmezésben *Normal Display* megjelenítési módban van, melyben a megjeleníthető karaktereket használhatjuk.

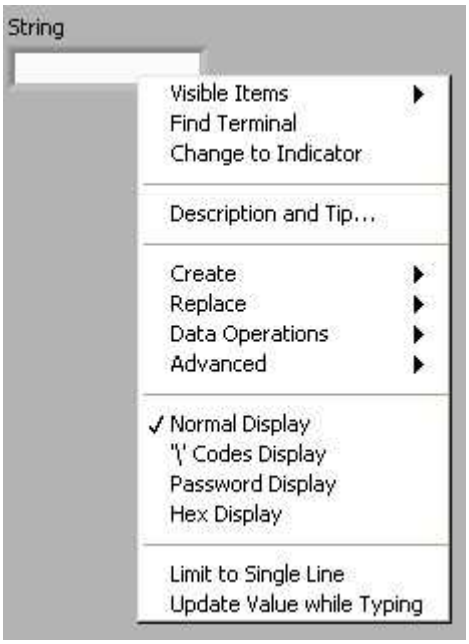
A *'\' Code Display* megjelenítési mód a nem látható karaktereket, mint például a "Backspace" (Visszalépés) és a "Tab" (Tabulátor) látható karakterekké alakítja. A láthatóságot a '\' karakterrel kezdődő kódolással biztosítja.

A *Password* módban a begépelt szöveg helyett a képernyőn csak * karakterek jelenjenek meg. Ezt a lehetőséget rendszerint jelszó megadásánál alkalmazzuk.

Hex Display módban karakterek helyett hexadecimális kódjuk látható a képernyőn.

A *Limit to Single Line* megjelenítési mód választása esetén a szöveges indikátorunkat egysoros kijelzőként használhatjuk.

Az *Update Value while Typing* menüpont kiválasztása esetén sztringünk rögtön frissítődik, amint begépeltük a karaktereket.

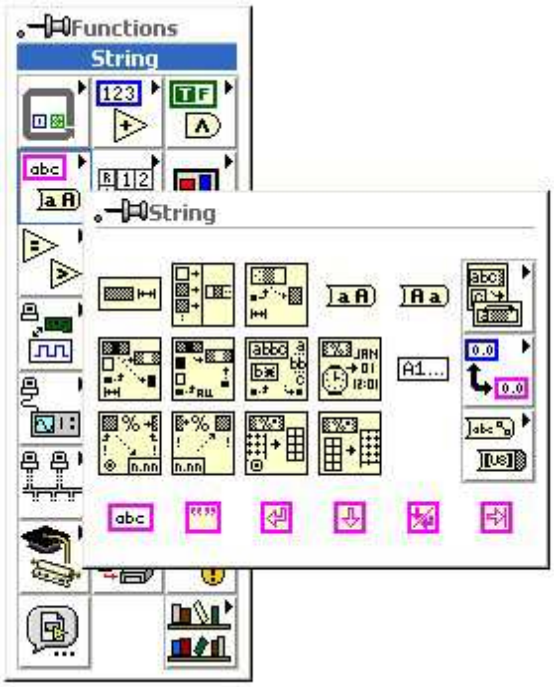


5.49. ábra. Szöveges elem saját menüje

5.9. Műveletek szöveges típusokkal

A sztring típusú elemekkel végezhető műveleteket a Functions paletta *String* tábláján találhatjuk *(5.50. ábra)*. Láthatunk itt számos szöveges adattal végezhető műveletet, különböző konverziós függvényeket, illetve konstansokat.

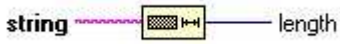
A konstansok a tábla alsó részén helyezkednek el, és lila színűek, ami egyébként a szöveges adatokra jellemző szín.



5.50. ábra. Szöveges elemek műveletei

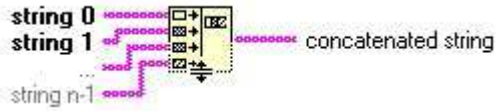
A szöveges műveletek közül itt a legfontosabbak, leggyakrabban használtak kerülnek tárgyalásra.

String Length (5.51. ábra): a bemenetre kötött szövegünk hosszát adja vissza



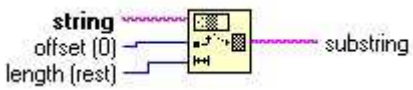
5.51. ábra. String Lenght művelet

Concatenate Strings (5.52. ábra): összefűzi a bemenetre kötött szövegrészeket. Mikor kiválasztjuk ezt a műveletet, alapértelmezés szerint két elemet fűzhetünk össze vele, de a Positioning Tool segítségével megnövelve a műveleti elem méretét, több bemenetünk is lehet.



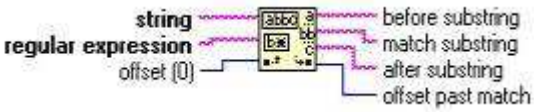
5.52. ábra. Concatenate Strings művelet

String Subset (5.53. ábra): egy szövegből *offset*-edik karaktertől *length* hosszúságú résszöveget választ ki.



5.53. ábra. String Subset művelet

Match Pattern (5.54. ábra): egy adott szövegben résszöveget keres, ez alapján kiolvashatjuk a kereset résszöveg előtti szövegrészt (before substring), utáni szövegrészt (after substring), a keresett résszöveget és a keresett résszöveg utáni karakterpozíciót (offset past match).



5.54. ábra. Match Pattern művelet

To Upper Case (5.55. ábra): a kisbetűket nagybetűre változtatja.

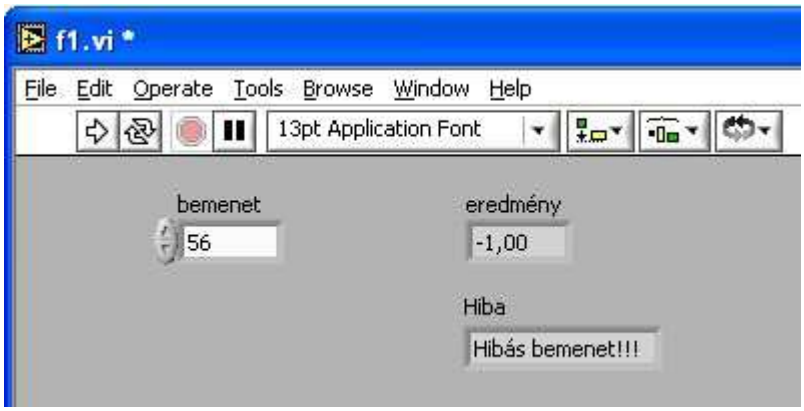
5.55. ábra. To Upper Case művelet

5.10. Feladatok

1. Készítsünk programot, mely beolvas egy egész számot 100 és 200 között, és azt ki is írja. Ha hibás az adatbevitel (‚nincs benne a tartományban a beolvasott érték), akkor írjunk ki hibaüzenetet, és az eredmény 1 legyen.

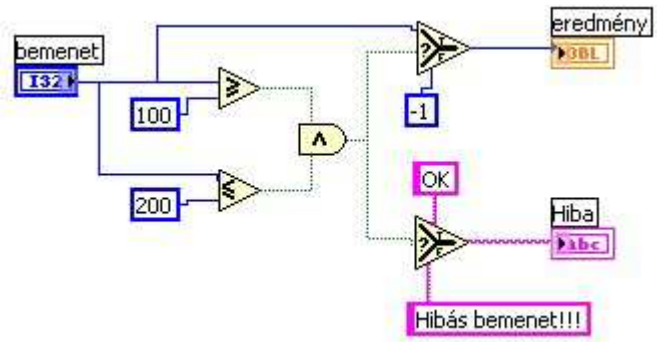
Megoldás:

Helyezzünk el a felhasználói felületen egy egész szám típusú vezérlő elemet és egy megjelenítő elemet, valamint egy szöveges megjelenítő elemet a hibaüzenetnek *(5.56. ábra).*



5.56. ábra. A program felhasználói felülete

Ezután lépjünk át a Diagram panelre. Meg kell vizsgálnunk a bemenetünket, hogy nem kisebb-e 100-nál, és nem nagyobb-e 200-nál. Ha mindkettő együtt teljesül, akkor az eredményünk a bemenet lesz, a hiba helyére írassunk ki OK feliratot, ha pedig valamelyik feltétel nem teljesül, akkor az eredményre 1-et kössünk, a hibára pedig valamilyen hibaüzenetet *(5.57. ábra)*.



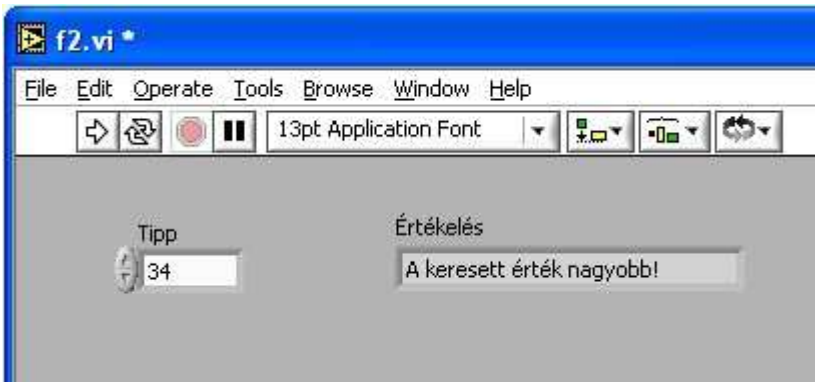
5.57. ábra. A program forráskódja a Diagram panelon

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

2. Készítsünk számkitalálós játékot! Programunk generáljon egy 0 és 100 közötti egész számot. A felhasználó tippel egy számot a numerikus vezérlőelemen keresztül, ezt a tippet hasonlítsa össze programunk a generált értékkel, és írassuk ki, hogy a keresett szám nagyobb, vagy kisebb, vagy eltaláltuk. Ha nem találta el, újra tippeljen a felhasználó egészen addig, míg ki nem találja a keresett számot.

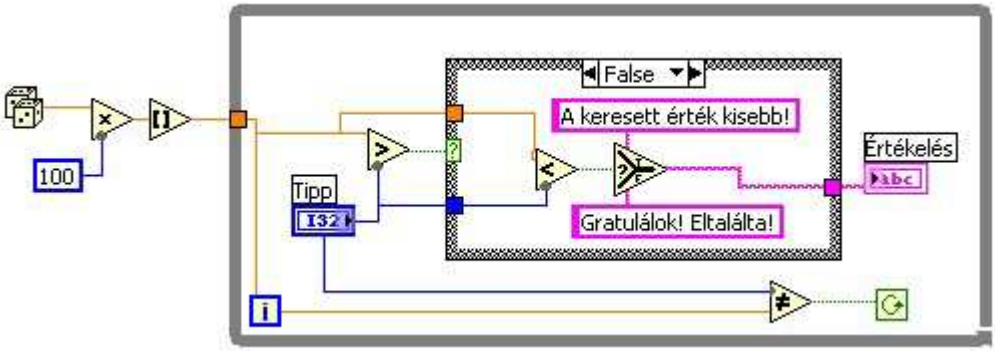
Megoldás:

Helyezzünk el a felhasználói felületen egy egész szám típusú vezérlő elemet a tipp beolvasásához, és egy szöveges megjelenítő elemet az üzenetek kiíráshoz *(5.58. ábra)*.



5.58. ábra. A program felhasználói felülete

Ezután lépjünk át a diagram panelre. Első lépés, hogy generáljunk egy véletlen számot 0 és 1 között, ezt szorozzuk meg 100-zal, és kerekítsük egészekre. Ha ez megvan, egy While ciklusban kell beolvasni a tippeket egészen addig, míg el nem találjuk a generált számot. A ciklus futása közben meg kell vizsgálnunk, hogy a generált szám nagyobb-e a tippnél, ha igen ezt írassuk ki. Ha nem nagyobb, akkor vagy kisebb, vagy egyenlő. Erre vonatkozóan is írassunk ki üzenetet *(5.59. ábra és 5.60. ábra)*.



5.59. ábra. A program forráskódja a Diagram panelon

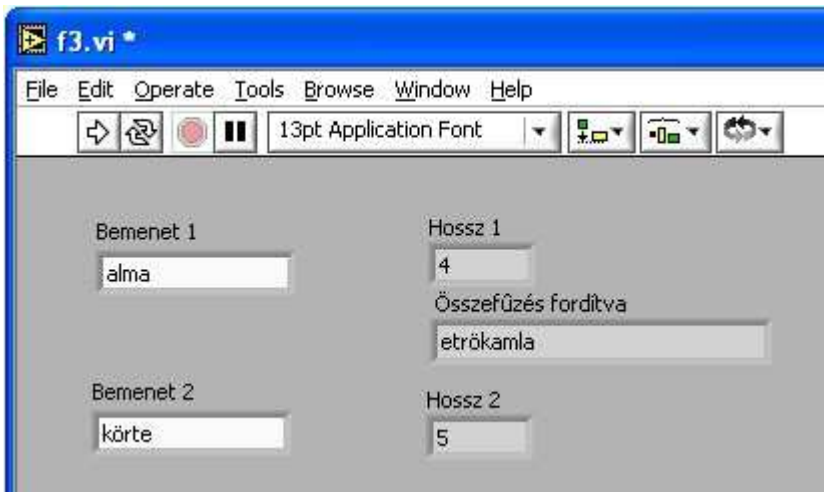
5.60. ábra. Az elágazás igaz értékhez tartozó lapja

Ha a ciklusban elhelyeztük a szükséges műveleteket és összekötöttük őket a megfelelő bemenetekkel, ill. kimenetekkel, lépjünk át a Front panelre, állítsuk be a bemenet kívánt értékét, és futtassuk a programot.

3. Készítsünk programot, mely beolvas két szöveges adatot, kiírja azok hosszát, összefűzi, és kiírja karakterenként fordított sorrendben!

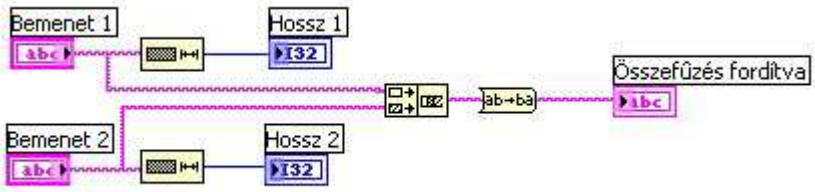
Megoldás:

Helyezzünk el a felhasználói felületen két szöveges vezérlő elemet a beolvasáshoz, egy szöveges megjelenítő elemet a fordított kiíráshoz, és két egész szám típusú megjelenítő elemet a szöveghosszak kiírására *(5.61. ábra)*.



5.61. ábra. A program felhasználói felülete

Lépjünk át a diagram panelre, és helyezzük el a szöveges műveletek közül a *String Length* műveletet, és másoljuk le, hiszen két példányban van rá szükségünk. Az összefűzéshez a *Concatenate Strings* műveletre lesz szükségünk, a szöveg megfordítását pedig a *Functions palette / String / Additional String Functions / Reverse String* műveletével érhetjük el *(5.62. ábra).*



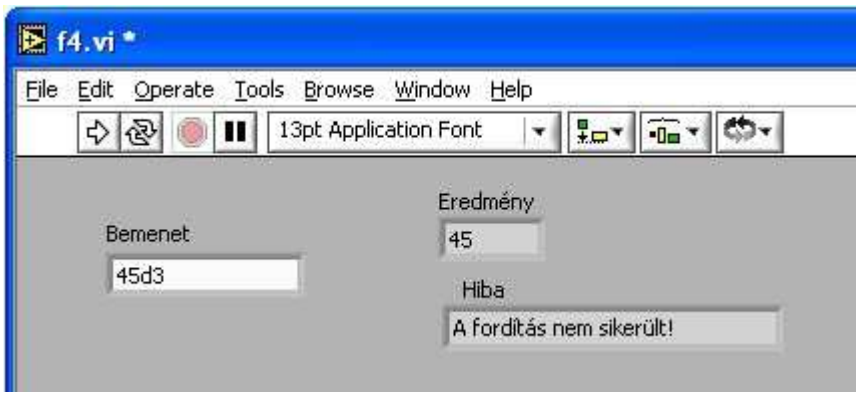
5.62. ábra. A program forráskódja a Diagram panelon

Ha elhelyeztük a szükséges műveleteket és összekötöttük őket a megfelelő bemenetekkel, ill. kimenetekkel, lépjünk át a Front panelre, állítsuk be a bemenetek kívánt értékeit, és futtassuk a programot.

4. Készítsünk programot, melyben egy szöveges elembe beolvashatunk egy számot, ezt konvertáljuk át egész számmá, ha lehetséges. Ha a beolvasott érték nem csak számjegyekből áll, írassunk ki hibaüzenetet!

Megoldás:

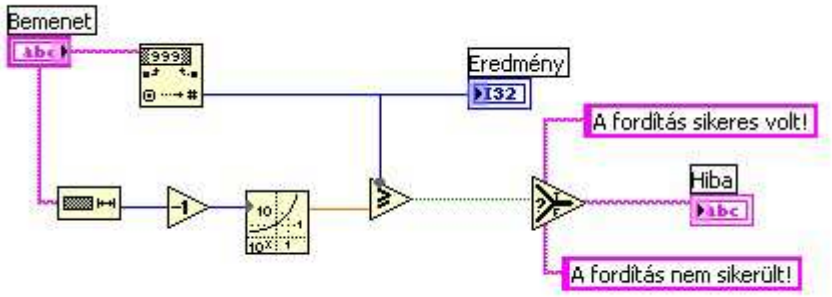
Helyezzünk el a felhasználói felületen egy szöveges vezérlő elemet a beolvasáshoz, egy szöveges megjelenítő elemet a hibaüzenetnek, és egy egész szám típusú megjelenítő elemet az eredménynek *(5.63. ábra).*



5.63. ábra. A program felhasználói felülete

Lépjünk át a diagram panelre, és helyezzük el a *Functions palette / String / String/Number Conversion* csoportjának *Decimal String to Number* műveletét. Ez a művelet végzi a

konverziót. Ha a szövegünk gond nélkül konvertálható, az eredmény megfelelő lesz, egyébként a konvertálás az első nem számjegyig történik. Mivel a művelet hibaüzenetet nem ad, nekünk kell kitalálni valamilyen ellenőrzési módot. Ha kizárjuk a szám 0-kal való kezdését, akkor elmondhatjuk azt, hogy helyes beolvasás esetén a konvertálás eredménye nagyobb vagy egyenlő lesz 10 a (szöveghossz-1)-edikennel. Pl.: 100 (3 hosszúságú) >= 10^2 *(5.64. ábra)*.



5.64. ábra. A program forráskódja a Diagram panelon

Ha elhelyeztük a szükséges műveleteket és összekötöttük őket a megfelelő bemenetekkel, ill. kimenetekkel, lépjünk át a Front panelre, állítsuk be a bemenet kívánt értékét, és futtassuk a programot.

5.11. Tömb és rekord típusú elemek

A tömb (Array) olyan típus, mely több azonos típusú elemet tartalmaz. Egy tömb lehet egy vagy több dimenziós. A tömb elemeit indexei segítségével azonosíthatjuk. Egy elem egyértelmű azonosításához annyi indexre van szükségünk, ahány dimenziós a tömbünk. A rekord (Cluster) típusú adatunk különböző típusú elemek halmaza. Tömb és rekord is, hasonlóan az egyszerű típusokhoz, lehet kontroll vagy indikátor.

A tömb és rekord típusú elemeket a Controls paletta *Array & Cluster* tábláján találhatjuk *(5.65. ábra)*. Ha elhelyezünk egy tömb vagy rekord típusú elemet a felhasználói felületen, még nem lesz egyértelmű, hogy milyen típusú elemek halmazát kívánjuk képezni. Tömböknél a Front panelen való elhelyezés után ki kell választanunk egy további elemet, mely a típust fogja szimbolizálni, és bele kell mozgatnunk a tömböt jelképező keretbe. Ekkor egy értékekkel nem rendelkező *egy dimenziós tömböt* (vektort) kapunk *(5.66. ábra)*.

5.65. ábra. Tömb és rekord típusú elemek

A tömb bal oldalán láthatjuk az index(ek) értékét, melynek számozása 0-tól kezdődik. Ha erre a kijelzőre kattintunk az egér jobb gombjával, akkor az index legördülő menüjét láthatjuk. Több dimenziós tömb előállításához ebben a menüben kell választanunk az *Add Dimension* menüpontot. Ha a Positioning Tool eszközzel átméretezzük a tömbünket, akkor látszik, hogy tényleg több dimenziós a tömbünk.

Ha a tömbünk kontroll módban van, akkor a felhasználónak kell feltöltenie elemekkel, ha indikátor módban van, akkor a programrészletünknek kell erről gondoskodnia.

5.66. ábra. Egy dimenziós tömb

Tömb feltöltésére általában a For ciklust használjuk, hiszen az esetek nagy részében ismerjük a tömbünk maximális indexeit *(5.67. (a) és (b) ábra)*. Ha For ciklussal töltjük fel tömbünket, akkor az indexelés automatikus lesz.

(a) (b)

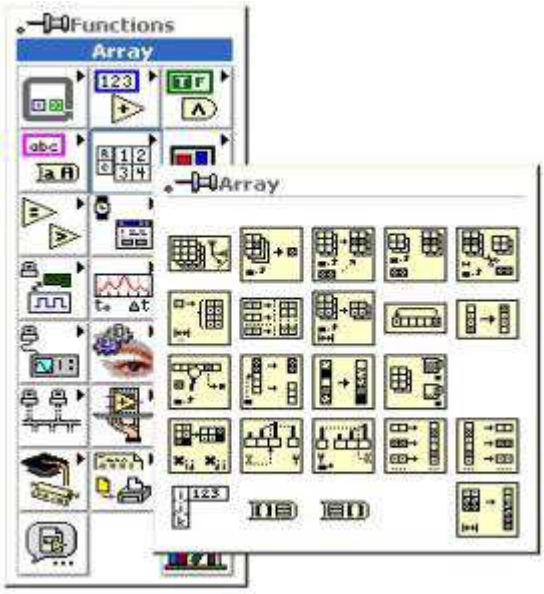
5.67. ábra. Tömb felépítése For ciklussal

Megjegyzés: Ha a ciklus lefutása után nem tömböt, hanem a legutolsó eredményt szeretnénk megkapni, akkor a ciklus kivezetési pontjának legördülő menüjében le kell tiltanunk az automatikus indexelést (Disable indexing).

Tömb elemei nem csak egyszerű elemek lehetnek. Tömb eleme lehet rekord, ekkor a tömbünk azonos felépítésű rekordok halmaza lesz. Ezzel szemben tömb eleme nem lehet tömb, tehát a LabVIEW a kétdimenziós tömböt nem egydimenziós tömbök tömbjeként kezeli!

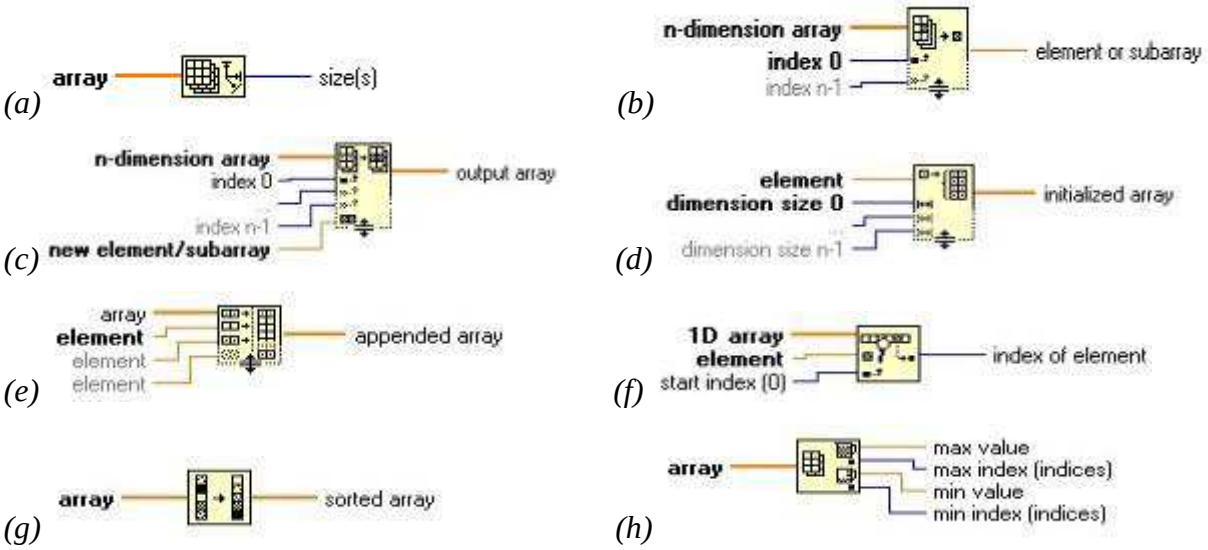
5.12. Műveletek tömbökkel és rekordokkal

A tömbökkel való műveletvégzés elemeit a Functions paletta *Array* tábláján találhatjuk *(5.68. ábra)*. Itt ezek közül csak a legfontosabbakat tárgyaljuk *(5.69. ábra)*.



5.68. ábra. Tömbök műveletei

Tömbökre alkalmazhatjuk az egyszerű műveleteket is, például összeadás, kivonás, stb., ekkor a művelet a tömb elemeire egyenként hajtódik végre.



(a) Array Size : Tömb méretének (elemszámának) meghatározása.

(b) Index Array : Index alapján elem kiválasztás a tömbből.

(c) Replace Array Element : Elem értékének kicserélése index alapján.

(d) Initialize Array : Tömb inicializálása (feltöltése azonos értékű elemekkel).

(e) Build Array : Új elemek beépítése a tömbbe.

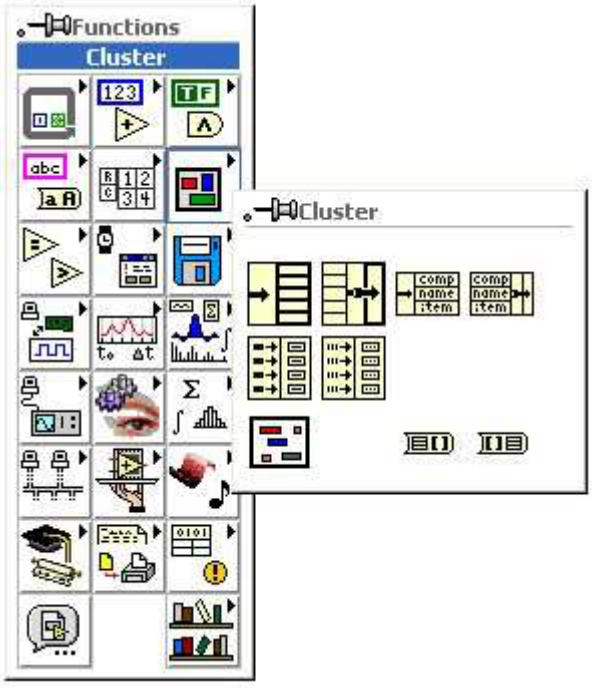
(f) Search 1D Array : Elem keresése egy dimenziós tömbben.

(g) Sort 1D Array : Tömb elemeinek sorbarendezése érték szerint.

(h) Array Max & Min : Maximum és minimum elem kiválasztása numerikus elemeket tartalmazó tömbből.

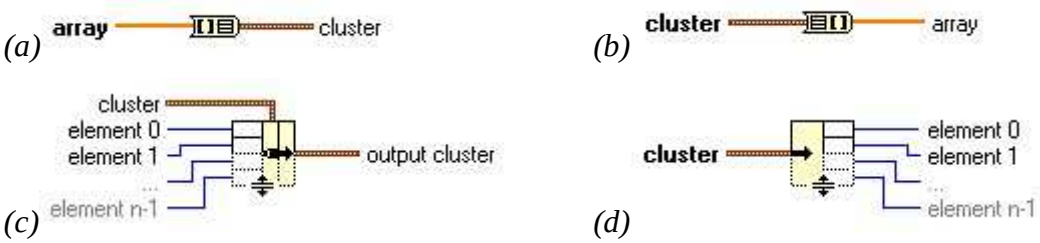
5.69. ábra. Tömbök fontosabb műveletei

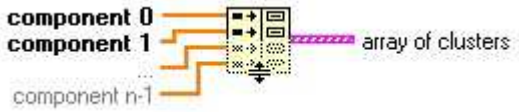
A rekordokkal végezhető műveletek elemeit a Functions paletta *Cluster* tábláján találhatjuk *(5.70. ábra)*. Ahogy a tömbök műveletei között megtalálhatjuk a tömb konstanst, itt megtalálhatjuk a rekord konstanst is, valamint mindkét függvénycsoportban fellelhetők a tömb-rekord, ill. rekord-tömb közötti konverziók.



5.70. ábra. Rekordok műveletet

Itt ezek közül a legfontosabbakat tárgyaljuk *(5.71. ábra)*.





(e)

(a) Array to Cluster : Tömb konvertálása rekorddá.

(b) Cluster to Array : Rekord konvertálása tömbbé.

(c) Bundle : Elemek összefűzése rekorddá.

(d) Unbundle : Rekord felbontása elemeire.

(e) Build Cluster Array : Rekordok tömbbe foglalása.

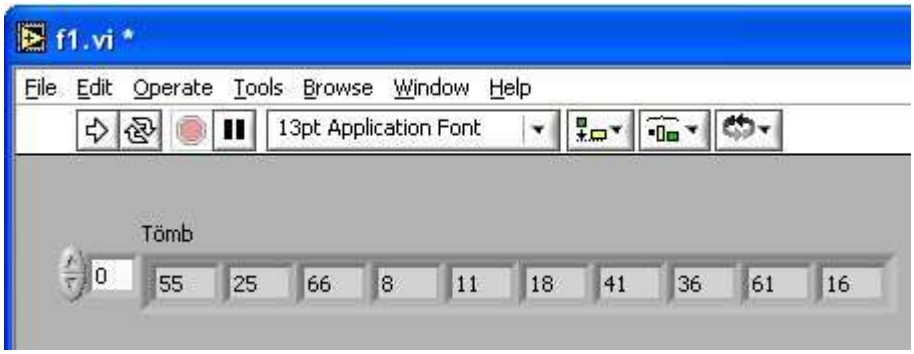
5.71. ábra. Rekordok fontosabb műveletei

5.13. Feladatok

1. Készítsünk programot, mely egy 10 elemű numerikus vektort tölt fel 0-100 közötti véletlen egész számokkal!

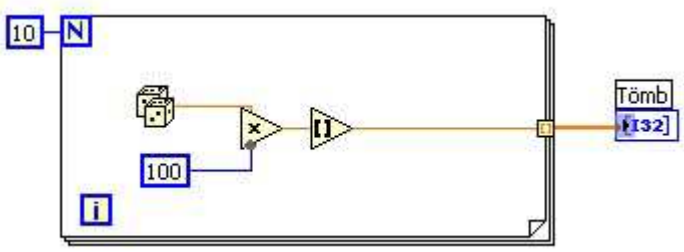
Megoldás:

Helyezzünk el egy tömb típusú elemet a felhasználói felületen, ebbe illesszünk egy numerikus indikátort, ezzel adjuk meg, hogy a tömb elemei szám típusú megjelenítő elemek lesznek *(5.72. ábra)*.



5.72. ábra. A program felhasználói felülete

Ezután lépjük át a Diagram panelre. Ha egy tömb elemeit szeretnénk elérni egyenként, akkor azt általában For ciklussal tesszük. Helyezzünk el egy For ciklust, és az N betűvel jelölt bemenetére kössünk egy konstans 10-est, ezzel meghatározzuk, hogy 10-szer fusson le a ciklusmag. A cikluson belül ki kell alakítanunk a 0-100 közötti véletlenszámokat. Ehhez el kell helyeznünk egy véletlenszám generátor műveletet, ennek eredményét meg kell szoroznunk 100-zal, majd egészekre kell kerekíteni. Mivel a ciklusoknak alaptulajdonsága, hogy képes tömböt készíteni a kivezetett adatokból, ezért a vektorunkat a cikluson kívül helyezve, és összekötve a 0-100 közötti véletlen egész számmal már el is végeztük a feladatot *(5.73. ábra)*.



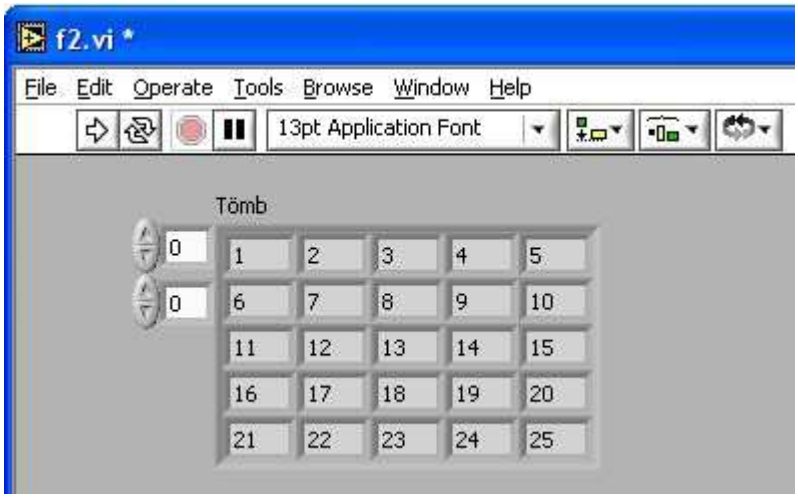
5.73. ábra. A program forráskódja a Diagram panelon

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

2. Készítsünk programot, mely egy 5x5-ös kétdimenziós tömböt tölt fel 1-től 25-ig egész számokkal balról jobbra, ill. fentről lefelé növekedve!

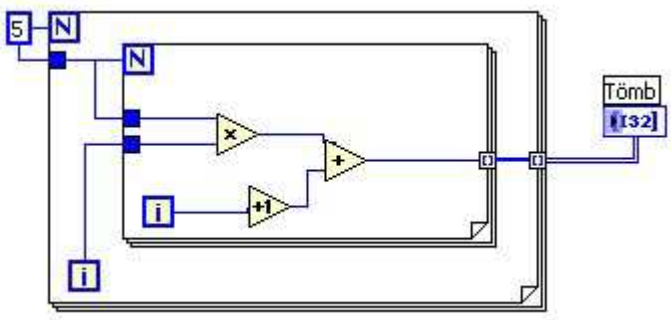
Megoldás:

Helyezzünk el egy tömb típusú elemet a felhasználói felületen, ebbe illesszünk egy numerikus indikátort. Ezzel egy egydimenziós tömböt hoztunk létre, akárcsak az előző feladatban. A tömb indexén kattintva az egér jobb gombjával, ott kiválasztva az *Add Dimension* menüpontot, megnövelhetjük tömbünk dimenzióinak számát. A tömb feltöltése az *5.74. ábrán* látható módon történjen meg.



5.74. ábra. A program felhasználói felülete

A kétdimenziós tömböt két egymásba ágyazott For ciklussal kezelhetjük, mindkettő ciklusnak 5-ször kell lefutnia. Karakteres programnyelvekben a cellák eredményét az i*5+(j+1) képlettel számolhatnánk ki, ahol i a külső, j pedig a belső ciklus változója, melyek értékei 0-ról kezdődnek. Ezt a képletet kell itt is megvalósítanunk, és hozzákötnünk a ciklusokon kívül álló tömbhöz *(5.75. ábra)*.



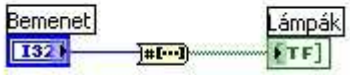
5.75. ábra. A program forráskódja a Diagram panelon

Miután kialakítottuk programunkat, lépjünk át a Front panelre és teszteljük programunk működését.

3. Készítsünk programot, mely a megadott decimális számot megjeleníti bináris alakban lámpák segítségével! A beolvasott érték ne legyen nagyobb 255-nél, így 8 lámpával megoldható a feladat. Ez a feladat már szerepelt a logikai és összehasonlító műveletek feladatai között, de most a megoldáshoz használjunk tömbműveletet!

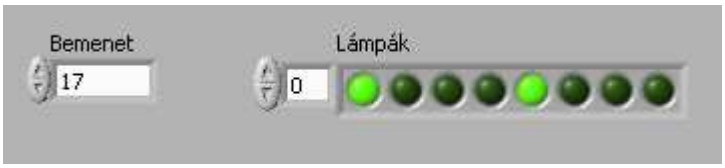
Megoldás:

A legegyszerűbb megoldás, ha a kijelző lámpasor egy logikai tömb, ekkor a dolgunk csak annyi, hogy a numerikus bemenetet logikai tömbbé konvertáljuk egyetlen művelettel *(5.76. ábra)*.



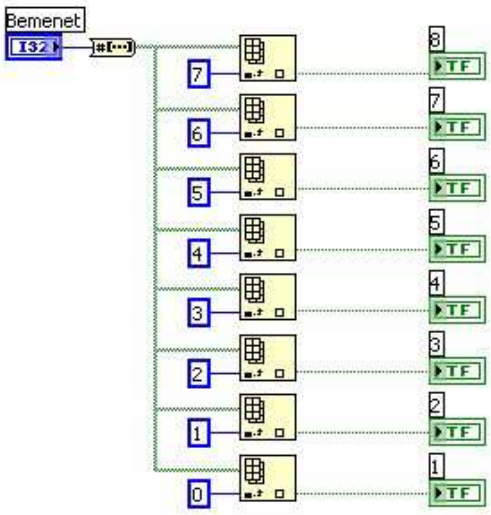
5.76. ábra. A program forráskódja a Diagram panelon

Ebben az esetben az eredményt a megjelenítő lámpasorról balról jobbra olvashatjuk ki *(5.77. ábra)*.



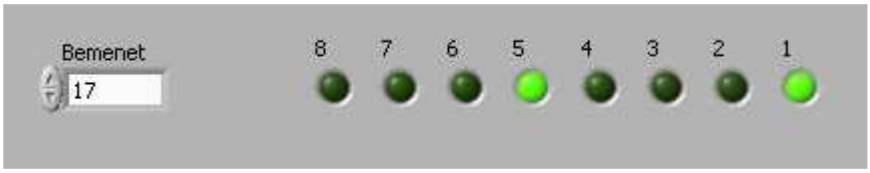
5.77. ábra. A program felhasználói felülete

Ha a lámpákat külön szeretnénk kezelni, és így akár jobbról balra megjeleníteni az eredményt, akkor a konverzió után a logikai tömböt elemeire kell bontani. A tömb elemeire bontását az *Index Array* művelettel végezhetjük el *(5.78. ábra)*.



5.78. ábra. A program forráskódja a Diagram panelon

Miután kialakítottuk programunkat, lépjünk át a Front panelre *(5.79. ábra)* és teszteljük programunk működését.



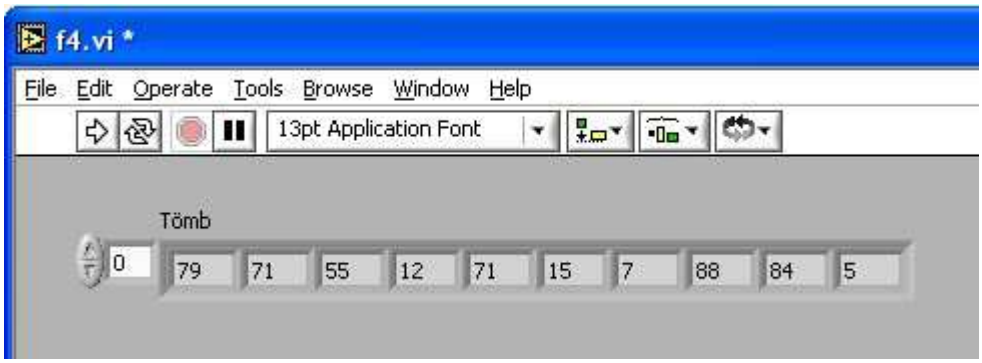
5.79. ábra. A program felhasználói felülete

4. Készítsünk programot, mely az 1. feladatban meghatározott egydimenziós tömböt úgy állítja elő, hogy nem használja ki a ciklus speciális lehetőségét, miszerint a kivezetett elemekből tömböt képez, hanem mi oldjuk meg a tömb felépítését!

Megoldás:

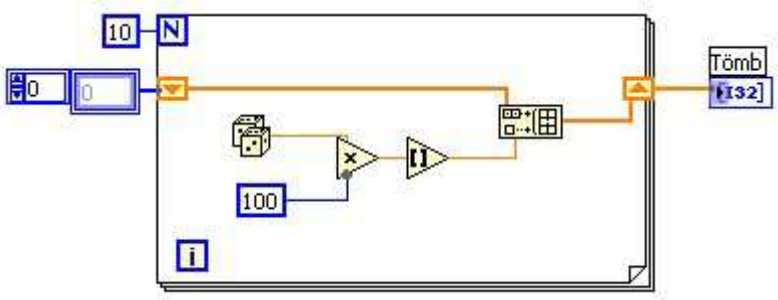
Előfordulhatnak olyan helyzetek, amikor nem tudjuk kihasználni ciklusaink tömbalkotó képességeit, ilyenkor magunknak kell megoldani ezt a feladatot.

Helyezzünk el egy numerikus indikátorokat tartalmazó tömböt a felhasználói felületen *(5.80. ábra)*, majd lépjünk át a Diagram panelre.



5.80. ábra. A program felhasználói felülete

A cikluson belül állítsuk elő a 0 és 100 közötti véletlen egész számot. A tömb építését a *Build Array* művelettel végezhetjük. A művelet első bemenete az a tömb legyen, melyhez hozzá szeretnénk fűzni egy újabb elemet, és ez az újabb elem a második bemenet. A hozzáfűzést az előző ciklusléptésben bővített tömbhöz szeretnénk megtenni, ehhez *Shift Register*-t kell használnunk. A legelső ciklusléptés előtt, meg kell adnunk a kiinduló értéket (üres tömb), és a ciklus lefutása után a végeredményt a regiszter jobb oldalából olvashatjuk ki *(5.81. ábra)*.



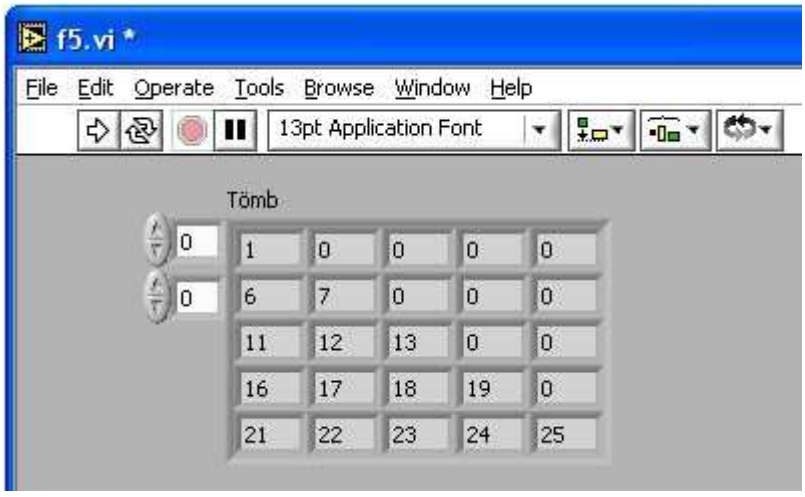
5.81. ábra. A program forráskódja a Diagram panelon

Miután kialakítottuk programunkat, lépjünk át a Front panelre és teszteljük programunk működését.

5. Készítsünk programot, mely a 2. feladatban előállított 5x5-ös kétdimenziós tömböt átalakítja úgy, hogy a sorindexénél nagyobb oszlopindexű elemek értéke 0 legyen!

Megoldás:

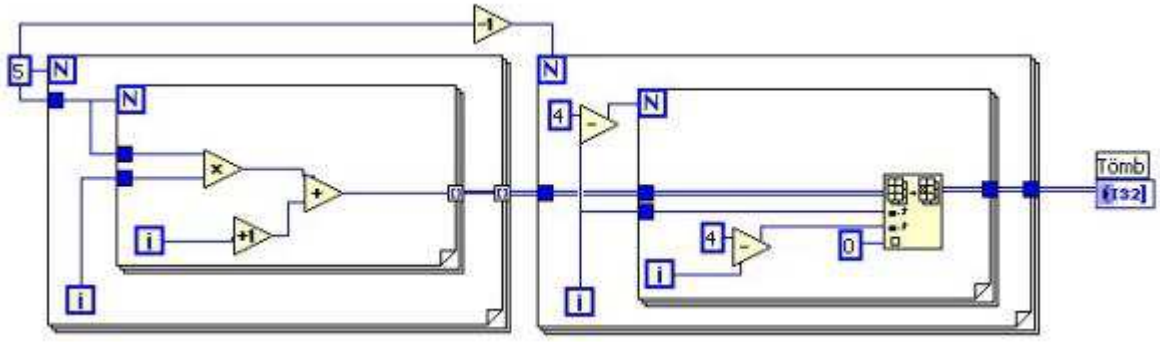
A tömb feltöltése az *5.82. ábrán* látható módon történjen meg.



5.82. ábra. A program felhasználói felülete

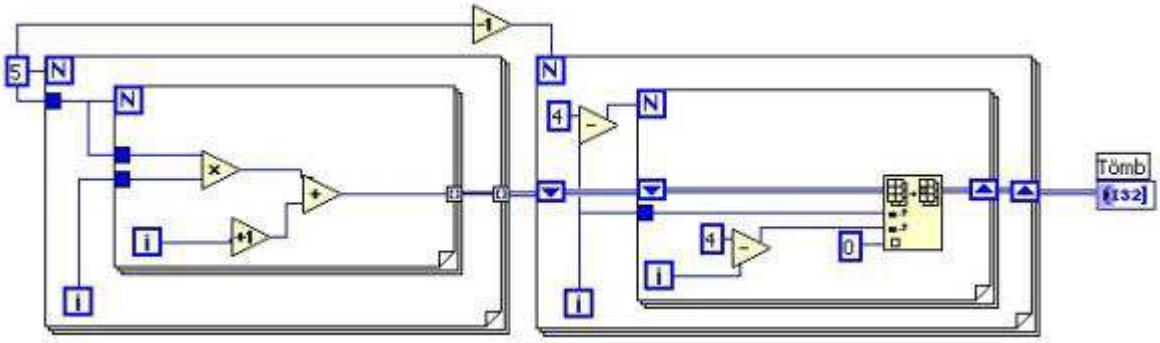
Tehát a tömbünk főátlója feletti elemeket 0-ra kell cserélnünk. Tömb elemét a *Replace Array Subset* művelettel cserélhetjük ki, melyet a belső ciklusba kell tennünk. Ha NxN-es az eredeti tömbünk, a külső ciklusnak N-1-szer kell lefutni, mivel az utolsó sorban nem kell cserélnünk. A belső ciklusnak N-1-i-szer kell lefutnia (, ahol i a külső ciklusváltozó), mivel az első sorban

4, a másodikban 3, aztán 2, majd 1 cserét kell megvalósítani. A cseréket a sor utolsó elemeitől kezdjük. Ha a cikluson belülre kötjük a tömböt, akkor a ciklus lebontja azt elemeire. Ezt a csatlakozópont pop-up menüjének Disable Indexing menüpontját választva kikapcsolhatjuk, mivel a cikluson belül is a tömböt akarjuk kezelni. Ha így futtatjuk a programot, látjuk, hogy nem a várt eredményt kapjuk *(5.83. ábra)*.



5.83. ábra. A program hibás forráskódja a Diagram panelon

A megoldásban az a hiba, hogy a cserét mindig az eredeti tömbön végezzük el, így csak az utolsó módosítás eredménye marad meg. Ezt a hibát *Shift Register*-ek segítségével küszöbölhetjük ki, így az előzőleg megváltoztatott tömbön hajtjuk végre az újabb módosításokat *(5.84. ábra)*.



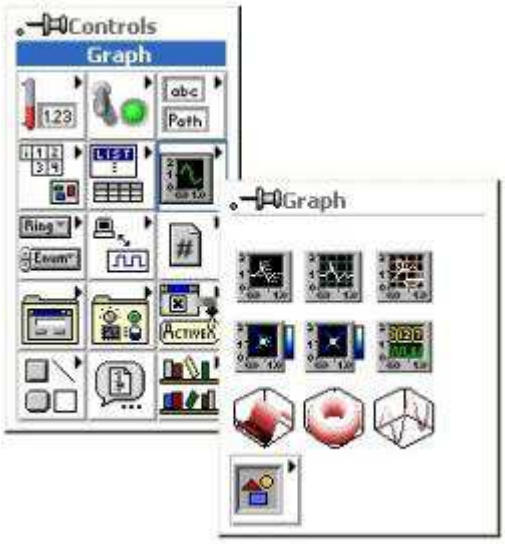
5.84. ábra. A program helyes forráskódja Shift Register-rel a Diagram panelon

Miután kialakítottuk programunkat, lépjünk át a Front panelre és teszteljük programunk működését.

5.14. Grafikus megjelenítők

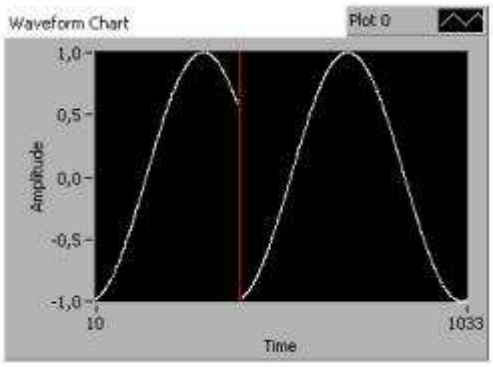
A grafikus kijelzők segítségével kétdimenziós vagy háromdimenziós grafikus formában jeleníthetünk meg eredményeinket.

A LabVIEW-ban a grafikus megjelenítők különböző típusait alkalmazhatjuk, melyeket a Controls paletta *Graph* tábláján találhatjuk *(5.85. ábra)*.



5.85. ábra. Grafikus megjelenítők palettája

Egyik ilyen megjelenítő a hullámforma diagram (Waveform Chart), mely alkalmas egy vagy több rajz egyidejű megjelenítésére *(5.86. ábra)*. A hullámforma diagram alkalmas folyamatos megjelenítésre (, ha cikluson belül helyezzük el), vagy tömb elemeinek index szerinti megjelenítésére.



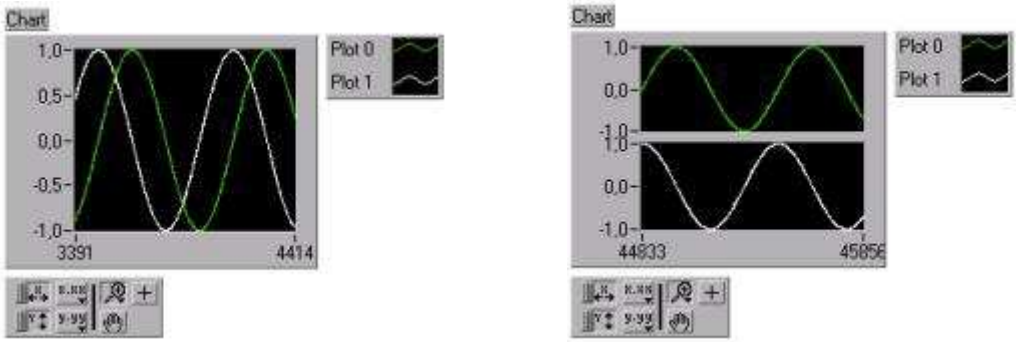
5.86. ábra. Waveform Chart

Megjeleníthetjük több adat értékeinek folyamatos változását, ilyenkor a megjeleníteni kívánt adatokat rekorddá kell összefűznünk (, melyet a Functions paletta Cluster tábláján található *Bundle* elemmel tehetünk meg).

Ha egy grafikus kijelzőt elhelyezünk a felhasználói felületen, automatikusan megjelennek a hozzá tartozó segédelemek is, melyen a felhasználó beállíthatja a megjelenítés különböző tulajdonságait (például kirajzolás színe, formája, vonaltípus, vastagság, tengelyek skálázása, stb.).

A grafikus kijelzőnk háromféle működési módban lehet, melyet a kijelző saját menüjének *Data Operations / Update Mode* menüjében választhatjuk ki. *Strip Chart* módban diagramunk a vízszintes tengely egy megadott méretű tartományban ábrázolható, ha ezt a tartományt túllépjük, akkor az ábránk továbbgördül. *Scope Chart* módban a tartomány túllépése esetén törlődik a diagramunk, és csak az új eredmények kerülnek ábrázolásra. *Sweep Chart* módban tartománytúllépés után egy függőleges vonal törli a régi adatokat, és mögötte láthatjuk az új értékeket.

Egyszerre több adat kirajzolása történhet a diagram egy ablakában (pop-up menü *Overlay Plots*) *(5.87. (a) ábra)*, vagy feloszthatjuk kijelzőnket (pop-up menü *Stack Plots*) *(5.87. (b) ábra)*.



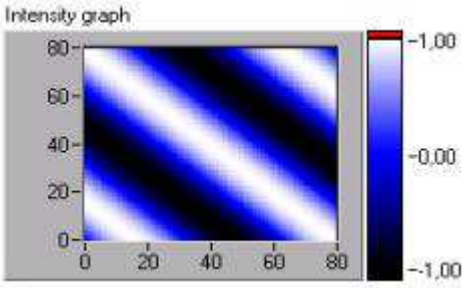
(a) Overlay Plots (b) Stack Plots

5.87. ábra. Waveform Chart megjelenítési lehetőségei

A hullámforma grafikon (*Waveform Graph*) nem képes az adatok folyamatos megjelenítésére, hanem egy vagy több tömb elemeit jeleníti meg.

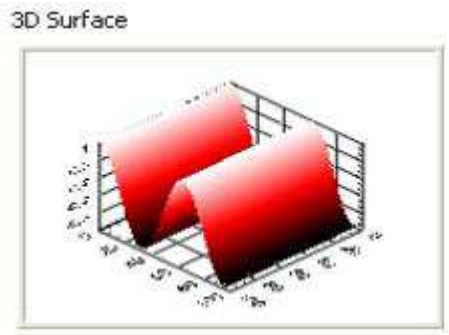
Az XY grafikon (*XY Graph*) két bemeneti tömb alapján rajzol ponthalmazt (X és Y koordináták alapján pontot rajzol ki).

Az *Intensity Chart* és *Graph* segítségével háromdimenziós adatot ábrázolhatunk kétdimenziós grafikonon *(5.88. ábra)*. Ez úgy történhet, hogy egy X, Y ponthoz egy harmadik értéket rendelünk, mely a színskála egy színét jelöli, így egy színes intenzitás grafikont kapunk.



5.88. ábra. Intensity Chart

Ugyanezt megtehetjük 3 dimenziós ábrázolásban a 3D Surface *(5.89. ábra)* és a 3D Parametric Surface segítségével, ahol kétdimenziós tömbök megadásával alakíthatunk ki térbeli felületeket.



5.89. ábra. 3D Surface

5.15. Feladatok

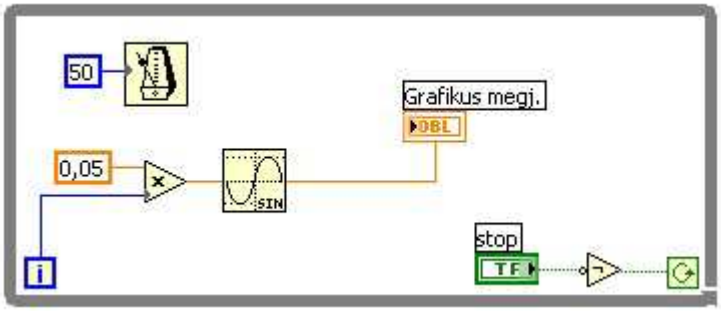
1. Készítsünk programot, mely egy grafikus megjelenítő elemen folyamatosan kirajzol egy szinusz-hullámot! A program addig fusson, míg a felhasználói felületre elhelyezett Stop gombot meg nem nyomjuk

Megoldás:

Helyezzünk el a Front panelen egy *Wavform Chart*-ot, mely egy bemenő adat időbeli (ciklusléptés függő) változását jeleníti meg, ill. egy Stop gombot.

A Diagram panelen alakítsuk ki ciklusunkat a Stop gombbal. Helyezzük a Chart szimbólumát a ciklus belsejébe, majd kössük rá a ciklusváltozó szinuszát *(5.90. ábra)*. Tulajdonképpen a feladattal így el is készültünk, a probléma csak az, hogy a görbénk a kijelző közepén elveszik, darabos az alakja, és nagyon gyorsan fut a jelünk a kijelzőn.

Az első hibára a megoldás, hogy az Y tengely értéktartományát automatikusra állítjuk. Kattintsunk a kijelzőn az egér jobb gombjával, és a válasszuk ki a pop-up menüből az *Y Scale/Autoscale Y* menüpontot. Hogy szép íves szinusz görbét kapjunk, ne egyesével növeljük a szinusz függvény bemenetét, hanem lassabban. Szorozzuk meg a ciklusváltozót például 0,05-dal, és ez legyen a szinusz függvény bemenete. Hogy szabad szemmel is követni tudjuk a változást, helyezzünk a ciklus belsejébe időkésleltetést (Time & Dialog / Wait Until Next ms Multiply, paramétere a késleltetés ideje milliszekundumban).



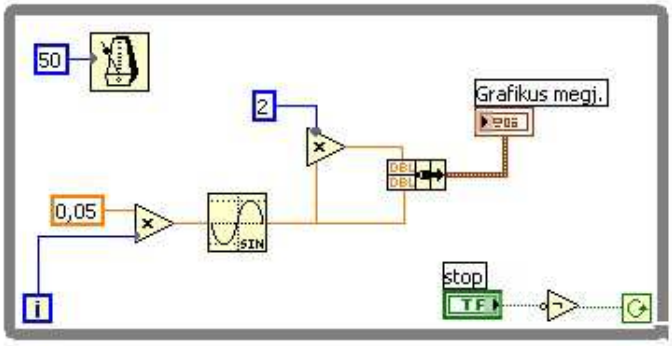
5.90. ábra. A program forráskódja

Miután kialakítottuk programunkat, lépjünk át a Front panelre és teszteljük programunk működését különböző megjelenítési módokban (pop-up menü *Advanced/Update mode* menü elemei).

2. Készítsünk programot, mely az 1. feladatban előállított szinusz-hullám mellett ugyanazon megjelenítőn kirajzolja a jel kétszeresét is!

Megoldás:

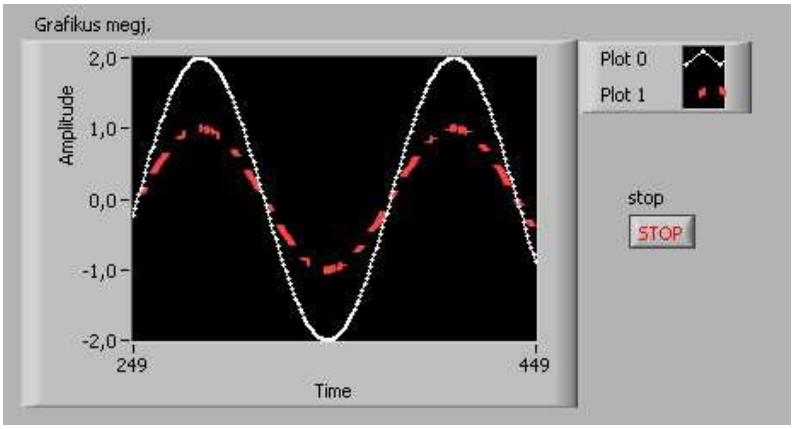
Ha egyszerre több görbét kívánunk megjeleníteni a Waveform Chart-on, akkor a bemenő adatokat *Cluster*-be kell fűznünk a *Bundle* művelettel, és a Bundle kimenetét köthetjük a megjelenítőre *(5.91. ábra)*.



5.91. ábra. A program forráskódja

Miután kialakítottuk programunkat, lépjünk át a Front panelre *(5.92. ábra)* és teszteljük programunk működését különböző kirajzolási módokban. Egyszerre több adat kirajzolása történhet a diagram egy ablakában (pop-up menü *Overlay Plots*), vagy feloszthatjuk kijelzőnket (pop-up menü *Stack Plots*).

A kirajzolás stílusát is tetszőlegesen állíthatjuk (vonaltípus, szín, vastagság, színezett terület, csak pontozott, stb.) a Chart mellett található kis ablakban. Ezeket a beállításokat egymástól függetlenül minden vonalra külön-külön megadhatjuk (Plot 0, Plot 1,).

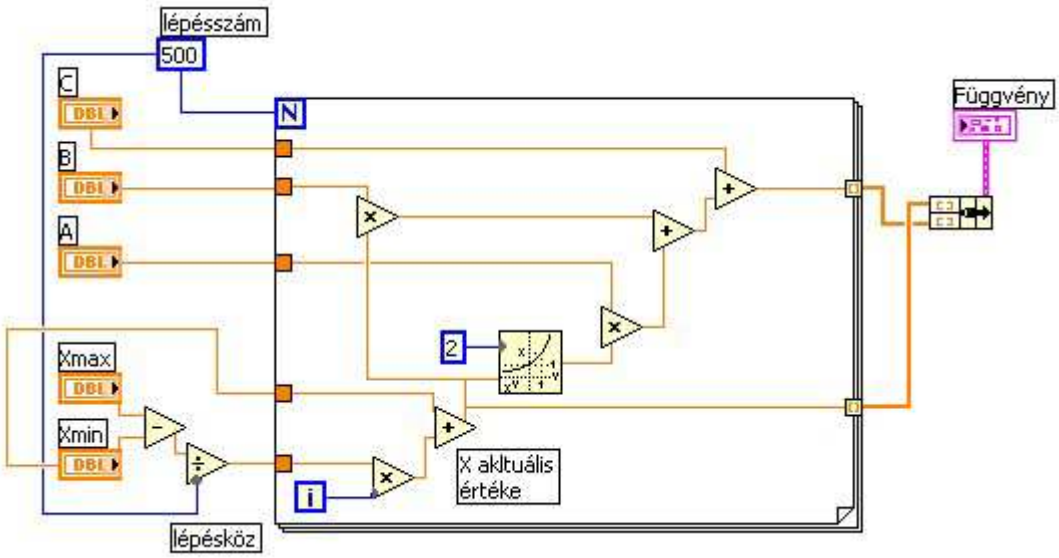


5.92. ábra. A program felhasználói felülete

3. Készítsünk programot, mely tetszőleges Xmin és Xmax tartományban, a felhasználó által megadott A, B, C paraméterekkel másodfokú függvényt ábrázol (, ahol $A*X^2+B*X+C=0$)!

Megoldás:

Ha adott értékek függvényében szeretnénk ábrázolni újabb értékeket, akkor azt az *XY Graph*-fal tehetjük meg. Tehát el kell helyeznünk a felhasználói felületen egy XY Graph megjelenítőt, és A, B, C, Xmin, Xmax beolvasására alkalmas vezérlőelemeket. El kell döntenünk, hogy milyen pontossággal ábrázoljuk a függvényt, azaz hány pontot számoljunk ki a tartományon belül (ez lesz a lépésszám). Ennek értéke legyen például 500. Így szép görbét kapunk kicsi tartományban is (pl.: 0 és 1 között is 500 pontot rajzolunk). Xmin, Xmax és a lépésszám alapján ki kell számolnunk a lépésközt (dX=(Xmax-Xmin)/ lépésszám). X aktuális értékét Xmin-től indítjuk, és mindig lépésközzel növeljük egészen Xmax-ig. Ezután már csak be kell helyettesíteni a képletbe. A kiszámított értékekből, ill. X aktuális értékeiből a ciklus segítségével tömböt képzünk, összefűzzük őket Cluster-ré és ábrázolhatjuk is az eredményt *(5.93. ábra)*.



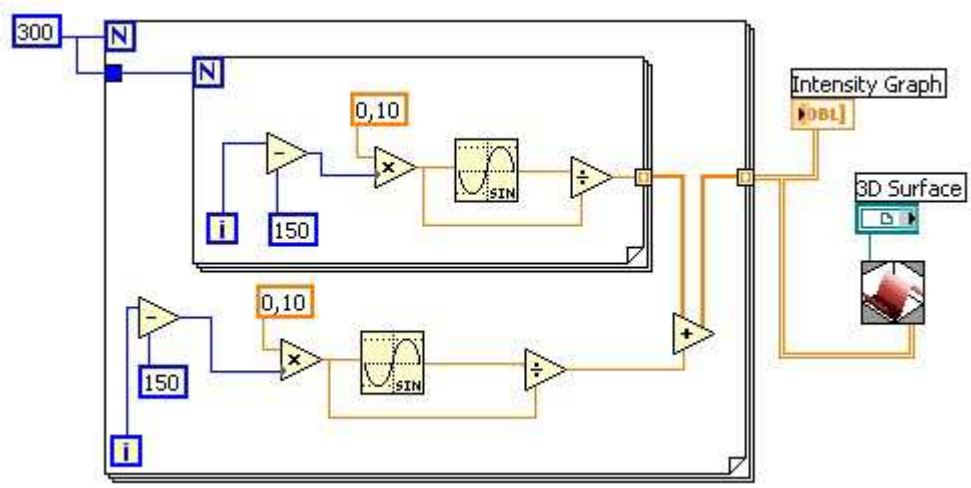
5.93. ábra. A program forráskódja

Miután kialakítottuk programunkat, lépjünk át a Front panelre és teszteljük programunk működését.

4. Készítsünk programot, mely háromdimenziós kijelzőn ábrázolja a $Z(x,y)=\frac{\sin(X)}{X}+\frac{\sin(Y)}{Y}$ függvény értékeit X=0 és Y=0 környezetében!

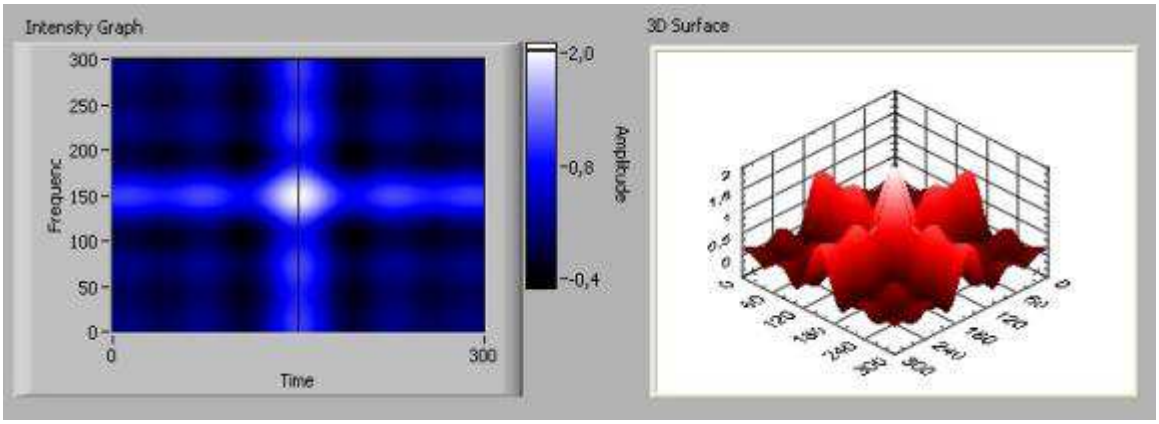
Megoldás:

Két egymásba ágyazott ciklusban a két ciklusváltozó segítségével adjuk meg X és Y értéket, melyet behelyettesítünk a képletbe. Majd kihasználva a ciklus tömbépítő képességét kössük az eredményt valamilyen 3D-s eredményt megjeleníteni képes kijelzőre *(5.94. ábra)*.



5.94. ábra. A program forráskódja

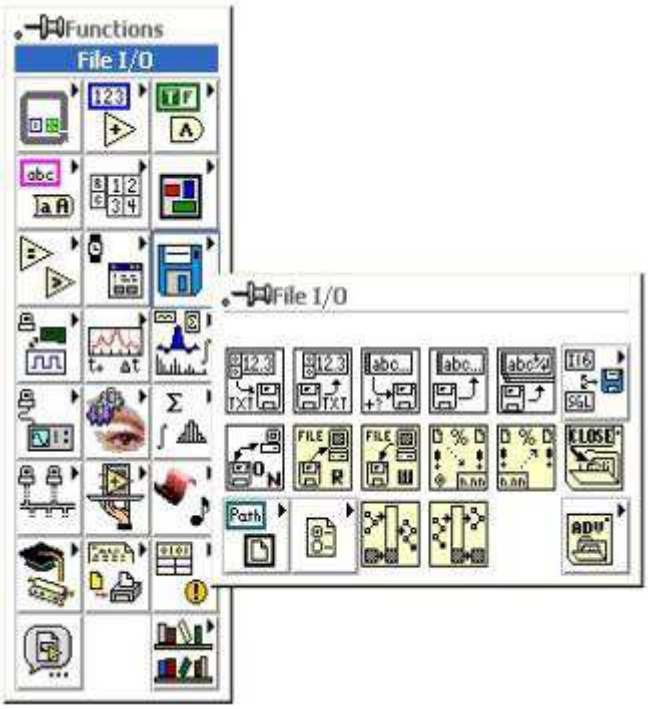
Miután kialakítottuk programunkat, lépjünk át a Front panelre és teszteljük programunk működését. Állítsuk az Intensity Graph tengelyeit automatikus skálázásúra. Az eredmény az *5.95. ábrán* látható.



5.95. ábra. A program felhasználói felülete

5.16. A fájlkezelés elemei

A fájlműveletek segítségével eltárolhatjuk adatainkat a háttértáron, majd ha szükségünk van rá, beolvashatjuk azokat. A fájlkezeléshez tartozó műveleteket a Functions paletta *File I/O* tábláján találhatjuk *(5.96. ábra)*. Egy fájlkezelési művelet három részből áll, megnyitjuk a fájlt (vagy létrehozzuk, vagy felülírjuk), utána írunk bele, vagy olvasunk belőle, majd bezárjuk.



5.96. ábra. File műveletek

A LabVIEW-ban három különböző hierarchia-szintű műveletet végezhetünk, *High-Level File Vis* (Magas szintű fájl műveletek), *Intermediate File Functions* (Középszintű fájlkezelő függvények) és *Advanced File Functions* (Haladó szintű fájlkezelő függvények).

A magas szintű fájl műveletek a *File I/O* tábla felső sorában találhatók, és középszintű fájlkezelő függvényekből tevődnek össze, melyek egy komplett feladat ellátására készültek, így megkönnyítik a programkészítő dolgát, viszont csak bizonyos feladatok ellátására alkalmas.

A középszintű fájlkezelő függvények a második sorban helyezkednek el, ezek felhasználása már több ismeretet igényel, viszont rugalmasan kezelhetőek, több lehetőséget nyújtanak a programozó számára.

Haladó szintű fájlkezelő függvények segítségével a LabVIEW fájl műveleteit alapelemekből építhetjük fel, és a legnagyobb rugalmasságot biztosítják a fájlok szervezésében.

Magas szintű műveletekkel könnyen elvégezhetjük a szöveges elemek fájlba írását, ill. fájlból olvasását. A művelet automatikusan megnyitja a kívánt fájlt, vagy ha ez nincs megadva, akkor egy ablakban választhatjuk ki, majd a fájlművelet után be is zárja az állományunkat. Ha hiba történik a művelet során, akkor egy dialógus ablakban jelzi ezt a műveleti elemünk.

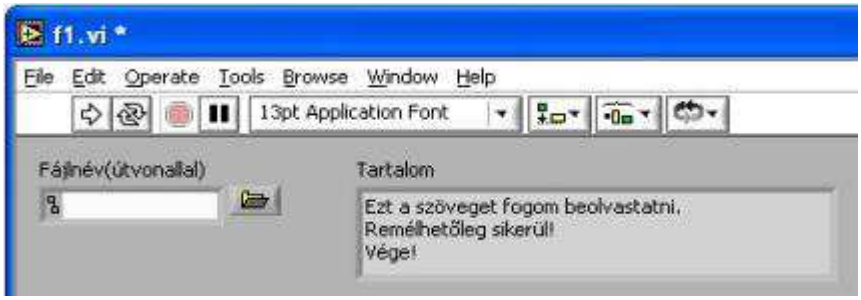
Hasonlóképpen magas szintű műveletként végezhetjük el kétdimenziós tömbök írását, olvasását illetve numerikus értékek írását, olvasását. Numerikus értéket tárolhatunk egész számként, vagy lebegőpontos számként.

5.17. Feladatok

1. Készítsünk programot, mely beolvassa a felhasználó által megadott szöveges állományt, és kiíratja a képernyőre!

Megoldás:

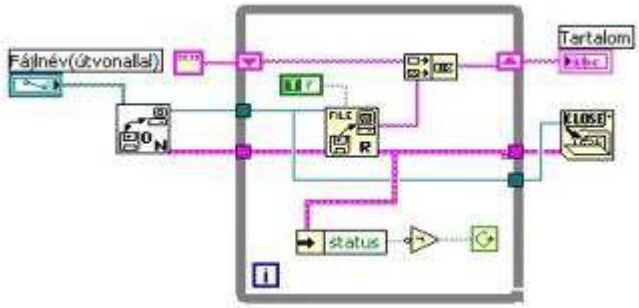
A felhasználói felületen el kell helyeznünk egy útvonal (path) vezérlő elemet, ahol a fájl nevét és helyét fogjuk beolvasni, valamint egy szöveges megjelenítő elemet, melynek méretét állítsuk eléggé nagyra, hogy jól látható legyen a kiíratás *(5.97. ábra)*, majd lépjünk át a Diagram panelre.



5.97. ábra. A program felhasználói felülete

A fájlok kezelése mindig a fájl megnyitásával kezdődik (*Open/Create/Replace File* művelet). Ezután olvashatunk a fájlból, írhatunk bele, kereshetünk benne. Ha mindent elvégeztünk, le kell zárni a fájlt (*Close File* művelet).

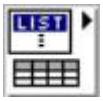
Nekünk olvasnunk kell a fájlból. Ezt soronként tesszük egészen a fájl végéig a *Read File* művelettel, mely hibajelet ad, ha nem tud tovább olvasni, azaz a fájl végére értünk. Ezt a hibajelet figyelve addig fusson a ciklusunk, amíg nincs hiba. A beolvasó művelet egyik bemenete a *soronkénti olvasási mód* (igen vagy nem) mely alapértelmezés szerint hamis, így nekünk egy igaz logikai konstanst kell bekötnünk. Mivel a sorokat külön-külön olvassuk be, ezeket össze kell fűznünk. Erre szolgál a *Concatenate Strings* művelet, és egy *Shift Register*-t is igénybe veszünk. A ciklus lefutása után lezárjuk a fájlt, ezzel készen is vagyunk *(5.98. ábra).*



5.98. ábra. A program felhasználói felülete

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

5.18. További kontrolcsoportok a Controls palettán



List & Table: Listaelemeket, és táblázat elemet választhatunk ebből a csoportból.

Ring & Enum: Itt szöveges és kép gyűrű elemet, menü elemet, valamint felsorolás típusú elemet találhatunk.


I/O: Ebben a csoportban a be- ill. kimenetekkel kapcsolatos elemeket találhatunk, mint például VISA, DAQ, IVI, Wavform, IMAQ.

Refnum: Ebben a csoportban referencia elemek közül válogathatunk.

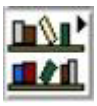

Dialog Contols: Itt megtalálhatjuk azokat az elemeket, melyek egy dialógusablak összeépítéséhez szükségesek lehetnek.


Classic Controls: A 3D-s elemek mellett itt megtalálhatjuk, és használhatjuk a korábbi verziókban megszokott 2D-s elemeket is.

ActivX: Programunkban ActivX elemeket is használhatunk, melyeket ebben a csoportban találunk.


Decorations: Itt díszítőelemeket találunk a felhasználói felület dekorálására.

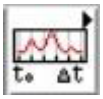

Select a Control ****: Vezérlőelemet a háttértárolóról is betölthetünk programunkba.


User Controls: Ha saját elemeket hozunk létre, azokat összegyűjthetjük a felhasználói elemek csoportjába. Ezeket az elemeket itt találjuk.

5.19. További függvénycsoportok a Functions palettán


Time & Dialog : Időzítéssel, dialógusablakokkal, és hibakezeléssel kapcsolatos függvények csoportja.

 *Data Acquisition*: DAQ eszközök kezelésének függvényei.

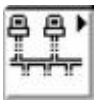
 *Waveform* : Waveform állományok kezelésének függvényei.

 *Analyze*: Jelfeldolgozással kapcsolatos, analizáló, és matematikai függvények csoportja.

 *Instrument I/O*: GPIB, VISA, és soros port kezelésének függvényei.

 *Motion & Vision*: Képfeldolgozás, mozgóképek kezelésének függvényei.

 *Mathematics*: Magasabb szintű matematikai műveletek és függvények.

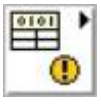
 *Communication*: Kommunikáció-, hálózatkezelő függvények csoportja.

 *Application Control*: Alkalmazáskezelő, menü- és súgókezelő függvények csoportja.

 *Graphics & Sound*: 3D-s grafikon-, kép- és hangkezelő függvények.

 *Tutorial*: LabVIEW Tutorial.

 *Report Generation*: Jelentéskészítés függvényei.

 *Advanced*: Rendszerközeli (haladó szintű) műveletek csoportja.

 *Select a VI*: Függvényeket, alprogramokat a háttértárolóról is betölthetünk programunkba.

 *User Libraries*: Ha saját függvényeket, alprogramokat hozunk létre, azokat összegyűjthetjük a felhasználói könyvtárakban.

5.20. Ellenőrző kérdések

1. Az elemek saját pop-up menüjéből mi az, ami kifejezetten a numerikus típusra jellemző?

2. Hol találjuk a numerikus elemekkel végezhető műveleteket?

3. Milyen működési tulajdonságai lehetnek egy logikai típusú elemnek?

4. Milyen adattípusok között végezhető összehasonlító művelet, és milyen típusú lesz az eredmény?

5. Milyen színnel jelöljük az egyes adattípusokat?

6. Milyen alapvető szöveges műveleteket ismerünk?

7. Hogyan hozunk létre tömböt a front panelen?

8. Hogyan képezünk többdimenziós tömböt?

9. Hogyan építhetünk tömböt egyszerűen ciklus segítségével?

10. Milyen működési módjai lehetnek egy grafikus megjelenítő elemnek?

11. Mi az alapvető különbség a *Chart* és a *Graph* között?

12. Hogyan tudunk egy grafikus megjelenítő elemen több ábrát kirajzoltatni?

13. Milyen jellegű fájlműveletekkel készíthetünk gyorsan, egyszerűen fájlból beolvasó, fájlba kiíró programot?

6. Lokális változók alkalmazása

Gyakran előfordul programkészítés során, hogy egy elemen kell megjelenítenünk olyan információt, melyhez a Diagram panelen nem elég egyetlen darab elem, és ha fizikailag másoljuk azt, akkor a Front panelünkön is megduplázódik a használt objektum. Ennek elkerülésére létrehozhatunk lokális változókat (*Local Variable*) melyek a kiválasztott elemünk értékével rendelkezik, de fizikailag nem jelenik meg a felhasználói felületen.

Lokális változó létrehozására többféle lehetőségünk van, megtalálhatjuk a *Functions* paletta *Structures* tábláján, és elhelyezhetjük a Diagram panelen, de ekkor még nincs értéke (),

nem használható. A változó saját menüjében a *Select Item* menüpontot választva megadhatjuk, hogy mely elemünkhöz tartozzon, mely elem értékét vegye fel *(6.1. ábra).*

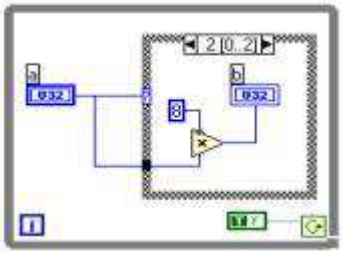
6.1. ábra. Lokális változó és saját menüje

A másik lehetőség, hogy megkeressük a Diagram panelen azt az elemet, melyhez lokális változót kívánunk létrehozni, és annak pop-up menüjében a *Create* menüpontján belül a *Local Variable* menüpontot választjuk. (Vigyázat, ha az elemünknek nincs saját címkéje (label), akkor a változónk ugyanúgy érték nélküli lesz, mintha a Functions palettából választanánk ki!)

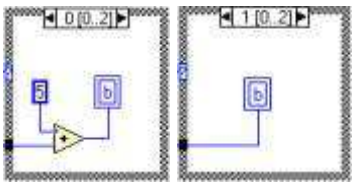
A lokális változó is lehet kontroll, vagy indikátor módban attól függően, hogy értékadásra, vagy kijelzésre használjuk. Éppen ebben rejlik használatának egyik oka, hiszen ha az elemünk kontrollként működik, akkor a hozzá tartozó lokális változó lehet indikátor, így a felhasználói felületen lévő elemünk egyben kijelző és vezérlő elemként is működik.

A másik lehetőség, amiért lokális változót használunk az lehet, hogy a diagramon az elemünk szimbóluma csak egy példányban van jelen, míg előfordul, hogy a különböző strukturális elemek által meghatározott "ablakok" közül nem csak egyben kell felhasználnunk ezt az elemet.

Nézzünk erre a problémára egy egyszerű példát! A feladat az, hogy helyezzünk el a felhasználói felületen két digitális elemet, egy kontrollt (a) és egy indikátort (b)! Indikátorunk értéke a kontroll értékétől függ, mégpedig úgy, hogy ha (a) értéke 0, akkor (b) legyen (a)+5, ha (a)=1, akkor (b)=(a), és ha (a)=2, akkor (b)=8*(a). A feladat egy lehetséges megoldását a *6.2. ábrán* láthatjuk.



ahol



6.2. ábra. A példa forráskódja

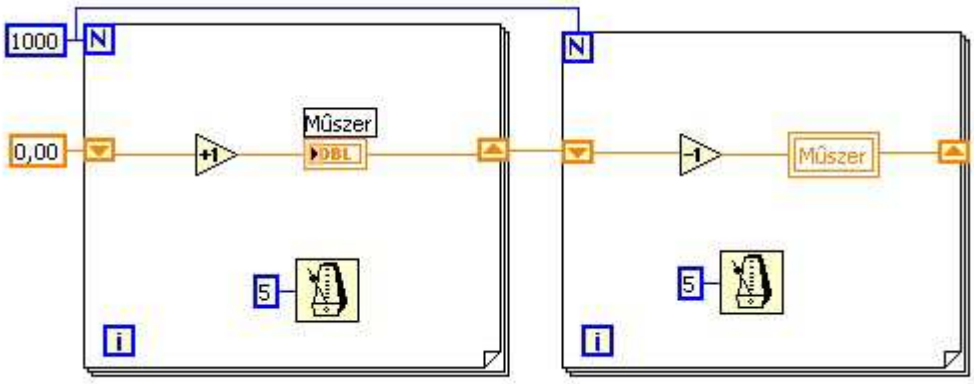
Megjegyzés: Ez egy egyszerűbb feladat, mely megoldásához akár nem is kellene lokális változót használnunk, hiszen (b) lehetne a CASE struktúrán kívül is, és ekkor a feltétel ágaiban csak a műveletek helyezkednének el. Ilyen módszerrel is tökéletes megoldást kapnánk, de akadhatnak olyan bonyolultabb feladatok, melyekhez mindenképp szükséges lokális változót használnunk.

6.1. Feladatok

1. Készítsünk programot, melyben egy mutatós műszer mutatóját kimozdítjuk 0-ról 10000-ig, majd visszafelé 0-ig!

Megoldás:

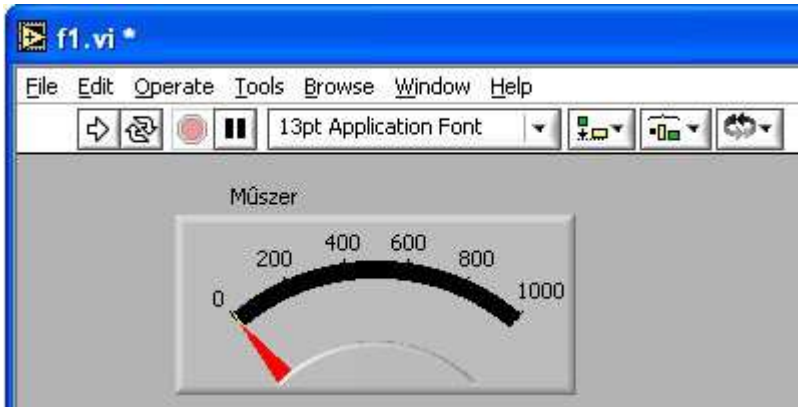
A 0-tól 10000-ig való mutató kibillentés egyszerű, egy For ciklusban folyamatosan növeljük a műszer értékét. Ezután a csökkentés történhet egy újabb ciklusban, de ekkor a műszer szimbólumát a második ciklusba is be kellene helyezni. Ha lemásoljuk a műszerünket, akkor a felhasználói felületen is kettő lesz belőle. Ha lokális változót hozunk létre, és ennek értékét csökkentjük, akkor a felhasználói felületünkön továbbra is csak egy elem marad. A lokális változó létrehozásának legegyszerűbb módja, hogy a műszer pop-up menüjéből a *Create/Local Variable* menüpontot választjuk *(6.3. ábra)*.



6.3. ábra. A program forráskódja

A feladat kicsit több gondolkodással megoldható lenne lokális változó nélkül is, de így egyszerűbb a dolgunk.

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését *(6.4. ábra)*.

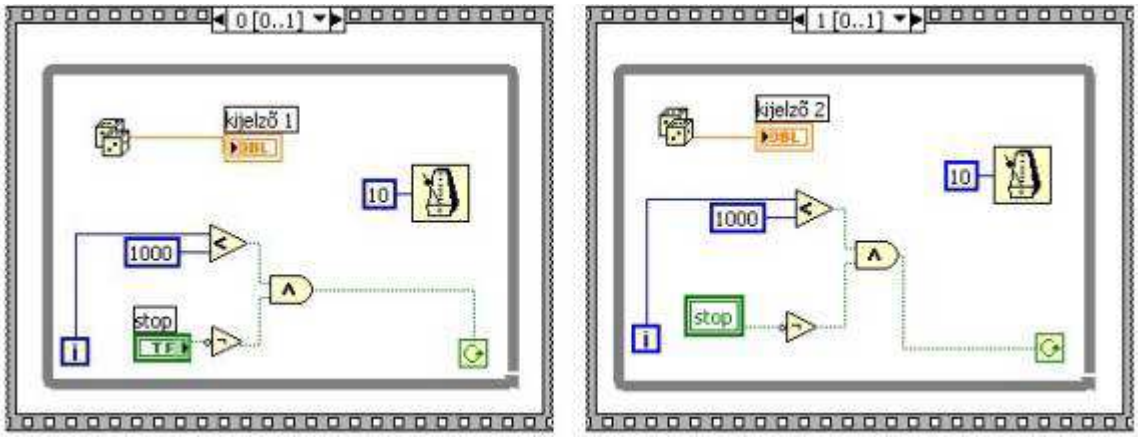


6.4. ábra. A program felhasználói felülete

2. Készítsünk programot, mely két Waveform Chart-on egymás után megjelenít 1000-1000 véletlen számot! Legyen egy Stop gombunk, mely segítségével a program futását bármikor megszakíthatjuk!

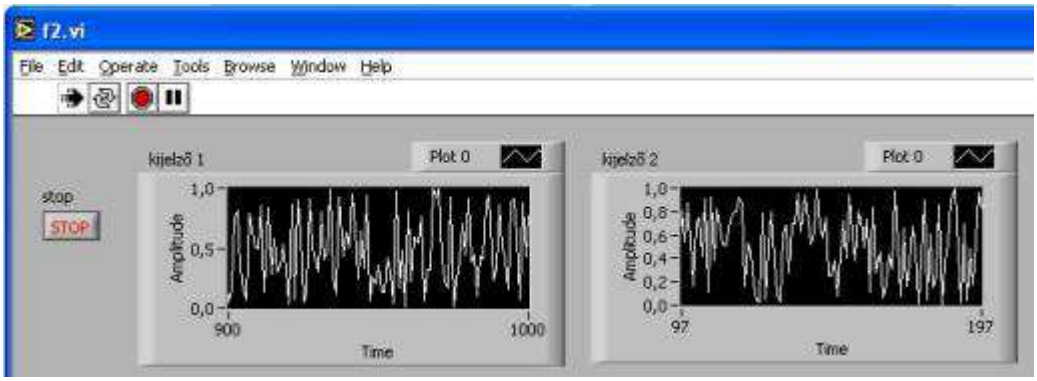
Megoldás:

Ha egymástól független, programrészleteket egymást követve szeretnénk futtatni, ahhoz szekvenciát kell alkalmaznunk. Ha bármikor le akarjuk állítani a program futását, akkor minden szekvencia ablak ciklusában meg kell jelennie a Stop gombnak, erre használjuk most a lokális változót *(6.5. ábra)*.



6.5. ábra. A program forráskódja

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését *(6.6. ábra)*.

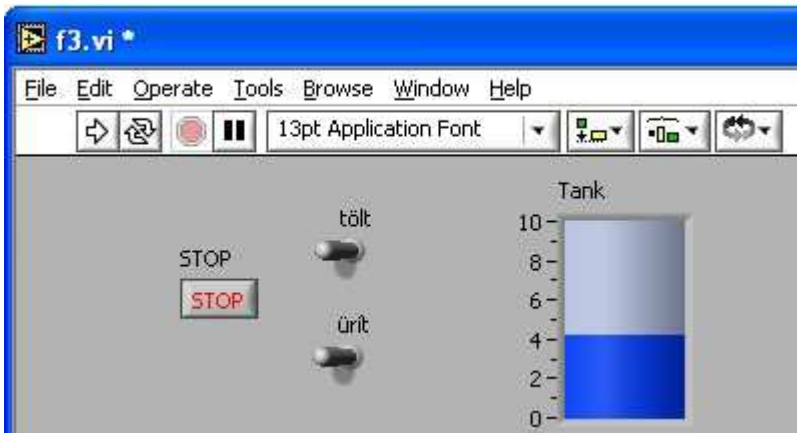


6.6. ábra. A program felhasználói felülete

3. Készítsünk programot, melyben egy tartályt tölthetünk, vagy üríthetünk egy-egy kapcsoló segítségével, és amint megtelt, vagy leürült a tartály, a megfelelő kapcsoló automatikusan kapcsoljon le!

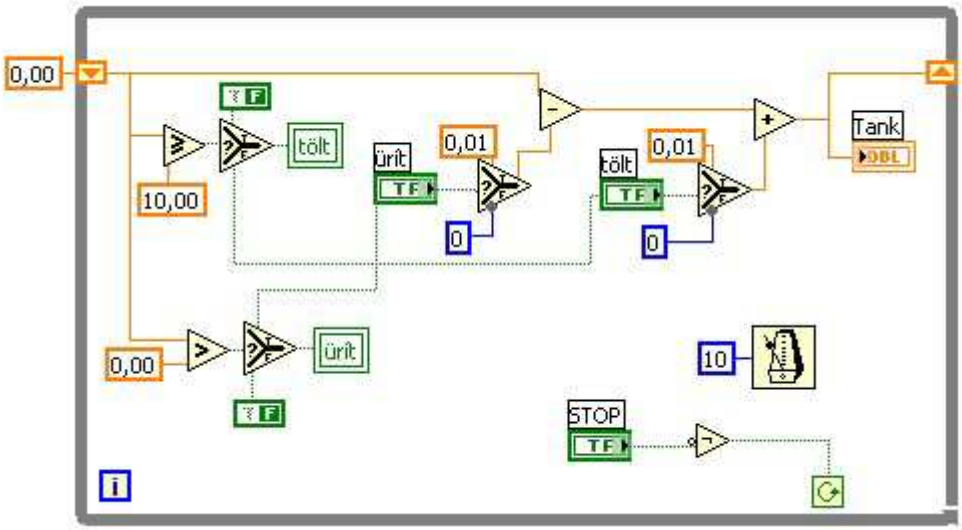
Megoldás:

Helyezzük el a szükséges elemeket a felhasználói felületen *(6.7. ábra)*, majd lépjünk a Diagram panelre.



6.7. ábra. A program felhasználói felülete

A tartály értékét folyamatosan növelni kell, ha a Tölt kapcsoló bekapcsolt állapotban van, ill. csökkenteni kell, ha az Ürít kapcsoló bekapcsolt állapotban van. (Ha a töltés és az ürítés azonos mértékű, akkor mindkettő bekapcsolása esetén azok kiegyenlítik egymást, és a tartály szintje nem változik.) Emellett figyelnünk kell az aktuális folyadékszintet, ha a tartály tele van, a Tölt kapcsolót le kell kapcsolnunk, és üres tartály esetén az Ürít kapcsolót kell kikapcsolni. Egyéb esetben a kapcsolók aktuális állásától függően történjen a működés. Mivel a kapcsolók vezérlőelemek, nem tudjuk a programmal befolyásolni értéküket, ezt csak akkor tehetnénk, ha megjelenítő elemek lennének. Ezért hozunk létre egy-egy lokális változót a kapcsolóknak, ezeket indikátorrá alakítjuk, és máris le tudja kapcsolni őket a programunk *(6.8. ábra)*.

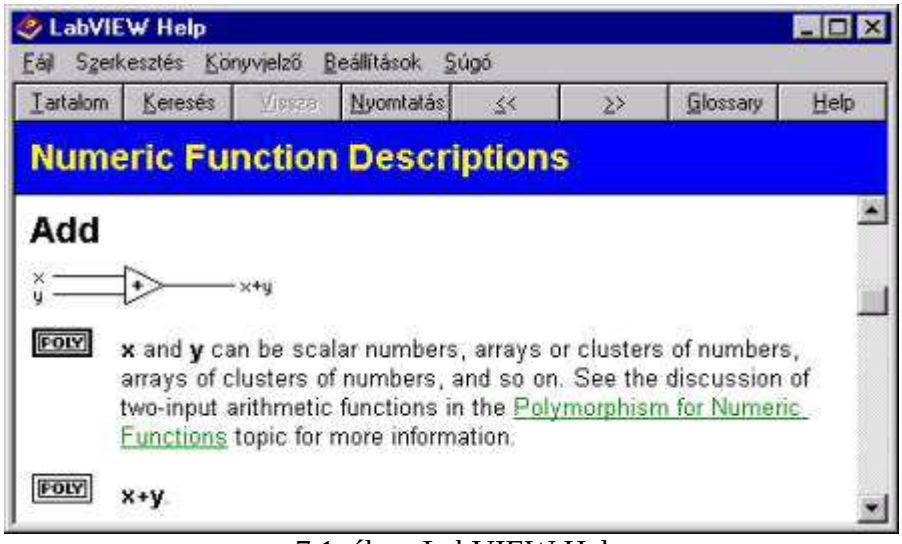


6.8. ábra. A program forráskódja

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

7. A Help használata

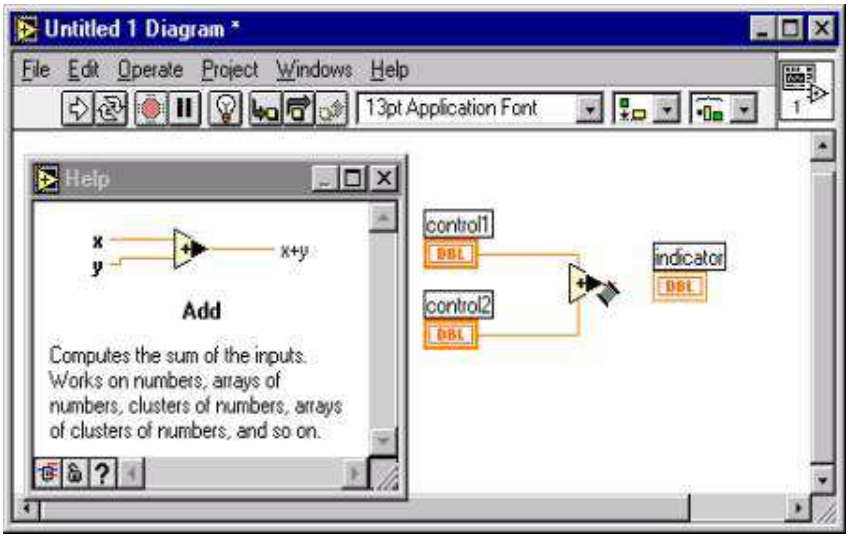
A LabVIEWban megtalálhatjuk a hagyományos súgó segítséget, mint általában a Windows alapú rendszereknél. Ezt a lehetőséget a *Help* menü *VI, Function, & How-To Help* menüpontjának választásával használhatjuk ki. Itt lehetőségünk van a témakörönkénti keresésre, de akár konkrét részeket is választhatunk *(7.1. ábra)*.



7.1. ábra. LabVIEW Help

Jól kihasználható lehetőség viszont az online help, melyet a LabVIEW készítői Contex Help-nek neveztek. Ha a Help menü *Show Context Help* menüpontját választjuk, egy kis ablak nyílik meg, ahol folyamatosan az éppen aktuális elemről kapunk információt, mely az elem működésére, és bekötésének módjára vonatkozik. Ez a lehetőség leginkább a műveleti elemeknél használható ki *(7.2. ábra)*.

Lehetőségünk van lock-olni, zárolni ( ) ezt a Help ablakot, ekkor nem az aktuális elemről, hanem a lockolás előtti elemről kapunk információkat egészen addig, míg fel nem oldjuk a zárolást.

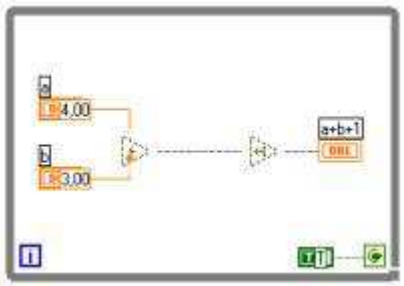


7.2. ábra. Contex Help használata

8. Hibaellenőrzés, programkövetési lehetőségek

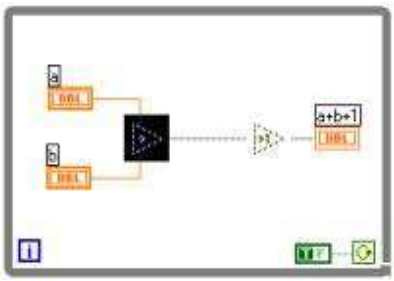
Ha programunk valamilyen hiba következtében nem futtatható, akkor az eszközsorban a *Run* (Futtatás) ikon helyén törött nyíl ( ) jelenik meg. Ha ekkor rákattintunk az ikonra, megjelenik egy hibalista, melyben felsorolva láthatjuk azokat a problémákat, melyek miatt programunk működésképtelen. A listában egy sorra kattintva az egér bal gombjával rövid leírást olvashatunk a hiba jellegéről. Kettőt kattintva, vagy a kiválasztást követően a *Find* gombot lenyomva megkereshetjük a hiba helyét a Diagram panelen.

Ha futtatható a programunk, de működése helytelen, nem a kívánt eredményt adja, akkor többféle lehetőségünk van a programkövetésre, hibaellenőrzésre. Az egyik ilyen lehetőség a Diagram panelen található *Highlight Execution* ( ), mely gomb megnyomásával láthatóvá válik az adatok "áramlása" elemről elemre a programszerkezetben, és minden elem kimenetén megjelenik az elemhez tartozó pillanatnyi kimeneti érték *(8.1. ábra).*



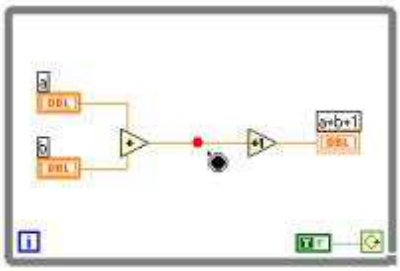
8.1. ábra. Highlight Execution

Programunk működésének ellenőrzésében nagy segítség lehet, ha a Diagram panelen elemről elemre haladva lépésenként futtathatjuk. Ezt a futtatási módot három gomb segíti, a *Step into* (), mely minden struktúrán és alprogramon belüli lépést egyenként hajt végre, a *Step over* (), mely a struktúrákat és alprogramokat egy elemnek tekinti, és az ott található algoritmust egy lépésben futtatja le, és a *Step out* (), mely lenyomásakor befejeződik a lépésenkénti futtatás *(8.2. ábra)*.



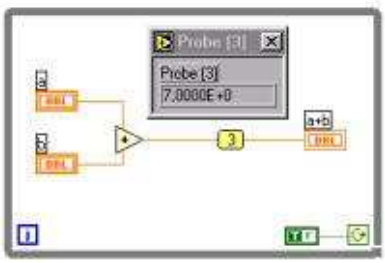
8.2. ábra. Lépésenkénti futtatás

Ha nem akarjuk ilyen részletességgel átvizsgálni egész programunkat, hiszen ez egy nagyobb alkalmazás esetén igen nagy feladat, akkor elhelyezhetünk töréspontokat (*Breakpoint*) a programunk kritikusabb, megvizsgálandó részein. Ezt a Tools paletta *Set/Clear Breakpoint* () eszközével tehetjük meg *(8.3. ábra)*.



8.3. ábra. Töréspont elhelyezése

Ha az egyes vezetékekben áramló adatok értékeire vagyunk kíváncsiak, hogy ellenőrizhessük algoritmusunk részeredményeit, elhelyezhetünk próba elemeket (Probe), mely egy kis ablakban folyamatosan kijelzi a vizsgált vezetéken átáramló értékeket. Próba elemet elhelyezhetünk a Tools paletta *Probe* () eszközével, vagy egyszerűen az egér jobb gombjával a vezetéken kattintva a legördülő menüből kiválaszthatjuk a Probe menüpontot *(8.4. ábra)*.



8.4. ábra. Probe elhelyezése

9. Elemek közötti összeköttetés hibalehetőségei

Az elemek közötti összeköttetésnek számos hibalehetősége van. Ezek közül a következők a legfontosabbak, melyek a programozás tanulási fázisában előfordulhatnak.

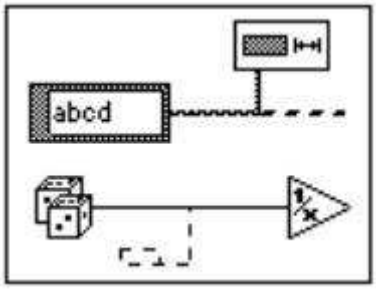
Eltérő típusok összekötése :

A két összekötött elem különböző típusú, például egy szám típusú kontroll értékét nem jelezhetjük ki egy logikai indikátoron *(9.1. ábra)*. (Ugyanez vonatkozik a műveletek kimenetén megjelenő jelekre is.)

9.1. ábra. Eltérő típusok összekötése

Maradék vezeték :

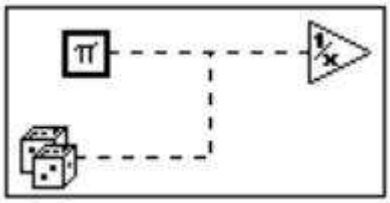
Ha elemeink összekötése után marad felesleges vezeték, akkor nem futtatható a programunk, tehát ezeket el kell távolítani *(9.2. ábra)*. Ezt megtehetjük a felesleges vezetékek egyenként történő kijelölésével, és törlésével, de a legegyszerűbb módszer erre a feladatra a *CTRL+B* billentyűk lenyomása, mely a program összes hibás bekötését egyszerre eltünteti.



9.2. ábra. Maradék vezeték

Több kontroll kapcsolódása :

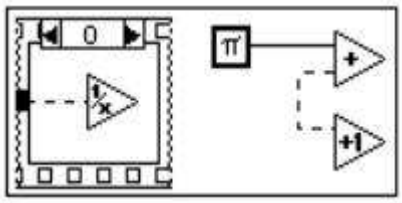
Egy kontroll értékét kijelezhetjük több indikátoron is, de egy megjelenítőelemnek nem adhat bemenetet egyszerre több vezérlőelem *(9.3. ábra)*!



9.3. ábra. Több kontroll kapcsolódása

Nincs forrás :

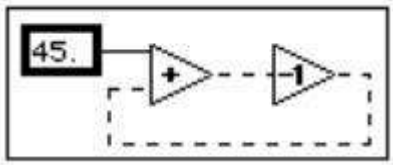
Ha egy műveleti elemnek nincs bemenete, akkor a programunk nem futtatható. Ha a bemenetet bekötöttük, akkor sem biztos, hogy használható adat jut a műveleti elemünkre *(9.4. ábra)*. Erre mindig oda kell figyelni!



9.4. ábra. Nincs forrás

Visszakötés :

Előfordulhat, hogy olyan algoritmust kell készítenünk, ahol a kimeneti értéket felhasználjuk a bemenetként *(9.5. ábra)*. Ekkor nem köthetjük vissza közvetlenül kimenetünket, mert ez hibát eredményez, hanem más módszert kell választanunk (például feladattól függően *Shift Register* használata ciklusban, vagy lokális változó használata).



9.5. ábra. Visszakötés

9.1. Ellenőrző kérdések

1. Mi az a lokális változó, mire használjuk?

2. Milyen segítségnyújtási lehetőséget biztosít számunkra a Help?

3. Milyen tesztelési lehetőségeket kínál a LabVIEW?

4. Hogyan tesztelnénk programunkat, ha az nagy mennyiségű adatot és műveletet tartalmaz, működése hibás, de tudjuk, hogy egy adott pontig biztosan helyes?

5. Mi a különbség a *Step into* és a *Step over* tesztelési módok között?

6. Mi történik, ha hibás programot futtatunk?

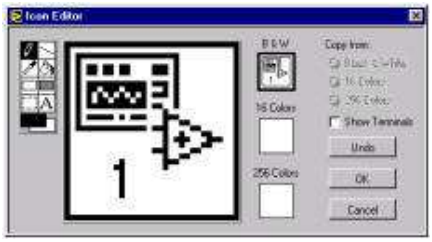
7. Mi lehet az oka annak, ha programunkban fekete szaggatott vezetéket találunk?

8. Hogyan törölhetjük ki felesleges, hibás vezetékeinket?

10. Alprogram (SubVI) kialakítása

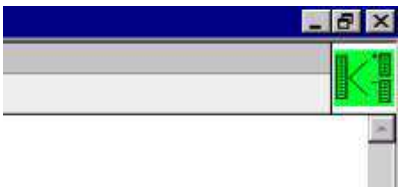
Elkészítet programunkat felhasználhatjuk más programok részeként, alprogramjaként is. A főprogramban alkalmazott SubVI ugyanolyan elemként jelenik meg, mint bármely más műveleti elemünk, konstansunk. A SubVI jelképe az általunk megrajzolt ikon lesz. Fontos tudnunk, hogy a LabVIEW működési sajátosságaiból eredően nem képes rekurzióra. Ha mégis szükségünk van ilyen alprogram alkalmazására, akkor ezt megírhatjuk C nyelven, és csatolhatjuk a VI-unkhoz.

Ikonnal minden program rendelkezik, az ablak jobb felső sarkában találhatjuk meg, és alapértelmezés szerint a képe:  . Ikonunk átalakításához annak legördülő menüjéből az *Edit Icon* menüpontot kell választanunk. Ekkor egy kis ablakban egy egyszerű rajzoló felület jelenik meg, ahol megrajzolhatjuk ikonunk képét *(10.1. ábra)*.



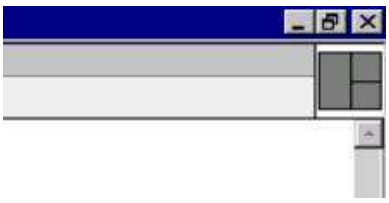
10.1. ábra. Ikonszerkesztő

Ha ezt megtettük és az OK gombra kattintottunk, akkor már meg is jelenik az ikon új képe az ablak jobb felső sarkában *(10.2. ábra)*.



10.2. ábra. Ikon megrajzolt képe

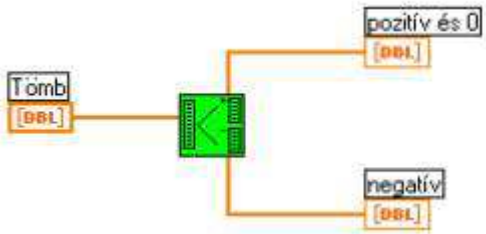
Ha programunkat alprogramként fel akarjuk használni, akkor definiálnunk kell a SubVI-nak átadandó és visszaadandó paramétereket. Ezt a konnektor kialakításával tehetjük meg. Az ikon legördülő menüjében a *Show Connector* menüpontot kiválasztva az ikon képe helyén a csatlakozási pontok helyeit láthatjuk *(10.3. ábra)*. Ha ezek száma, vagy elhelyezkedése nem felel meg számunkra, akkor az ikon legördülő menüjében a *Pattern* menüpontból számtalan kialakítás közül választhatunk.



10.3. ábra. Csatlakozási pontok

Ha kiválasztottuk a megfelelő konnektort, akkor már csak meg kell határoznunk, hogy melyik csatlakozó melyik paramétert fogja jelenteni. Ez úgy történik, hogy rákattintunk a konnektor egy csatlakozópontjára, majd a programunk egy elemére. Ha ez az elem kontroll, akkor a csatlakozópontba bemenetet lehet majd kötni, ha indikátor, akkor kimenő paramétert köthetünk hozzá. Így sorra meghatározva a bemenő és kimenő paramétereket már alkalmas lesz programunk arra, hogy SubVI-ként működjön.

Alprogramunkat úgy illeszthetjük más program Diagram paneljére, hogy a főprogram Functions palettából a *Select VI* lehetőséget választva betöltjük elkészített alprogramunkat *(10.4. ábra)*. Az ikonra kettőt kattintva megnyithatjuk SubVI-unkat.



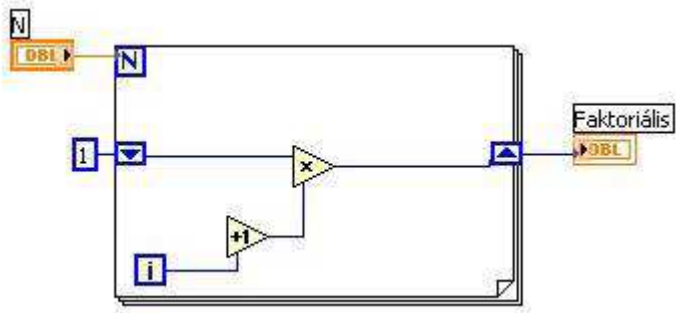
10.4. ábra. Alprogram használata

10.1. Feladatok

1. Készítsünk programot, mely kiszámítja a felhasználó által megadott egész N-ig a faktoriálisok (N!) összegét! Készítsük el a faktoriális kiszámítására alkalmas programot, és ezt alprogramként használjuk fel a feladat megoldásában!

Megoldás:

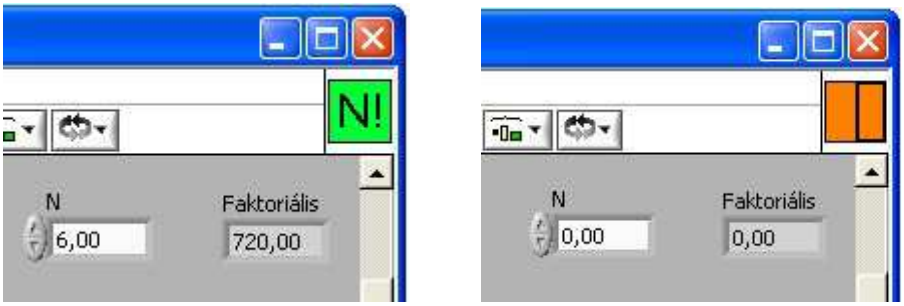
A faktoriális kiszámításához szükségünk van egy numerikus vezérlőre (N) és egy megjelenítőre (N!). 1-től N-ig szorozzuk össze az egészeket egy N-szer lefutó ciklusban Shift Register segítségével *(10.5. ábra).*



10.5. ábra. A program forráskódja

Miután a programmal elkészültünk, az ikon és a konnektor kialakítása következhet. Kattintsunk az egér jobb gombjával az ablak jobb felső sarkában található ikonon, és válasszuk ki az *Edit Ikon* menüpontot. A megjelenő egyszerű rajzolóprogrammal készítsük el a program ikonjának képét.

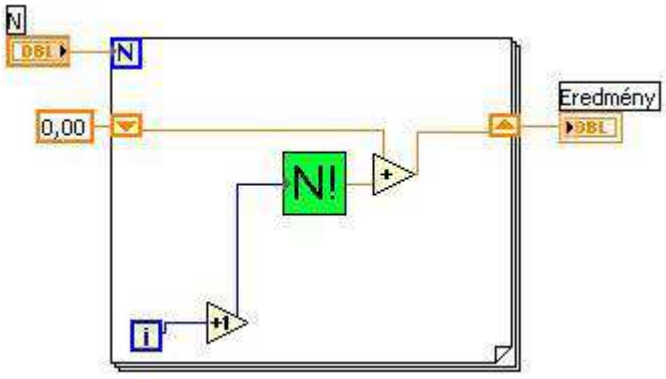
Ezután alakítsuk ki a program bemeneteit, ill. kimeneteit. Szintén kattintsunk jobb gombbal az ikonon, és a *Show Connector* menüpontot választva megjelennek a bemeneti és kimeneti terminálok *(10.6. ábra).* A terminálra kattintva a bal egérgombbal, azután ugyanígy kattintva egy elemen már össze is rendeltük az elemet a csatlakozóponttal. A vezérlő elemek bemenetek, a megjelenítő elemek kimenetek lesznek.



10.6. ábra. A program felhasználói felülete (Ikon és Connector)

Az elkészített programunkat elmentve, majd beillesztve egy másik programba úgy viselkedik, mint egy hagyományos függvény.

Készítsük el a külső programot, ahol összegezzük a faktoriálisokat 1-től N-ig. Ehhez szintén For ciklusra és Shift Registerre lesz szükségünk. Az alprogram beillesztése a Functions paletta *Select a VI* menüjének választásával történik *(10.7. ábra).*



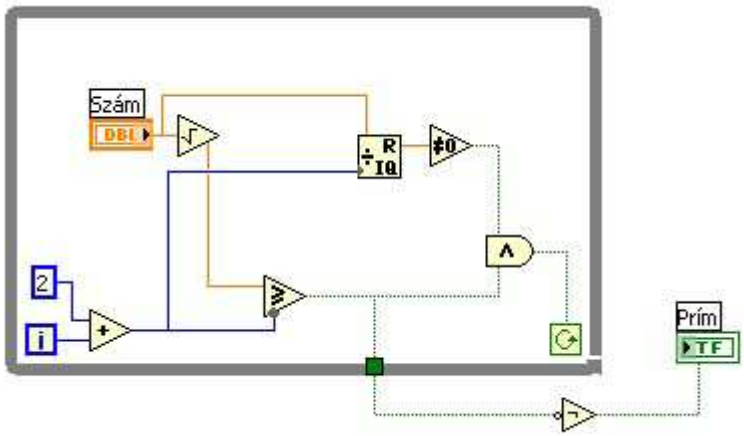
10.7. ábra. Alprogram alkalmazása

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

2. Készítsünk programot, mely a felhasználó által megadott egész N-ig összegyűjti a prímszámokat egy tömbbe! Készítsük el a prímszám vizsgálatra alkalmas programot, és ezt alprogramként használjuk fel a feladat megoldásában!

Megoldás:

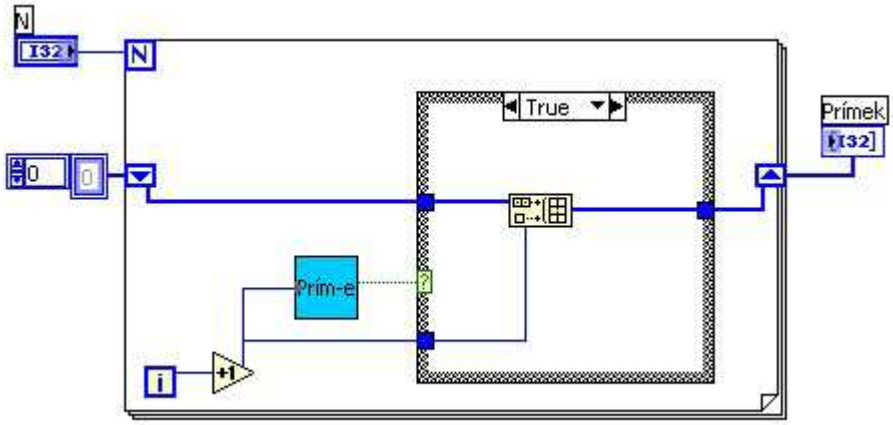
Egy számot akkor nevezünk prímszámnak, ha csak 1-gyel és önmagával osztható. Tehát, ha 1-től a számig egyesével lépkedve elosztjuk a ciklusváltozóval a számot, akkor csak két osztója lehet. Ezt optimalizálva 2-től szám-1-ig ha találunk osztót, akkor a szám nem prím. Tovább optimalizálva, ha 2-től a szám gyökéig nem találunk osztót, akkor biztosan prím a szám *(10.8. ábra).*



10.8. ábra. A prím program forráskódja

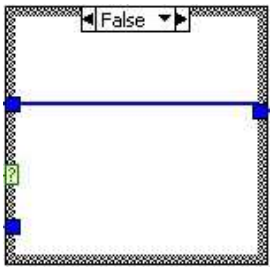
Ha kész a program, alakítsuk ki az ikont és konnektort!

Alakítsuk ki a főprogramunkat! 1-től N-ig (N a beolvasott szám) vizsgáljuk meg a számokat, hogy prímek-e. Ha igen, helyezzük őket tömbbe *(10.9. ábra).*



10.9. ábra. A lprogram használata

A tömb folyamatos bővítéséhez *Shift Registerre* és a *Build Array* műveletre van szükségünk. A tömb növelését az alprogram eredményétől függően egy *Case* struktúra igaz ágában hajtjuk végre. A hamis ágban nem növeljük a tömböt, viszont üresen nem hagyhatjuk, mert be kell kötnünk a struktúra kimeneti csatlakozópontját. A megoldás az, hogy a tömböt jelképező vezeték belép a struktúrába, és változatlanul ki is lép *(10.10. ábra).*



10.10. ábra. A szelekció hamis lapja

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

11. Saját elem készítése, elem átalakítása

A LabVIEW-ban lehetőségünk nyílik a kontrollok és indikátorok átalakítására, szerkesztésére. Így saját elemeket készíthetünk, melyeket más programjainkban is felhasználhatunk.

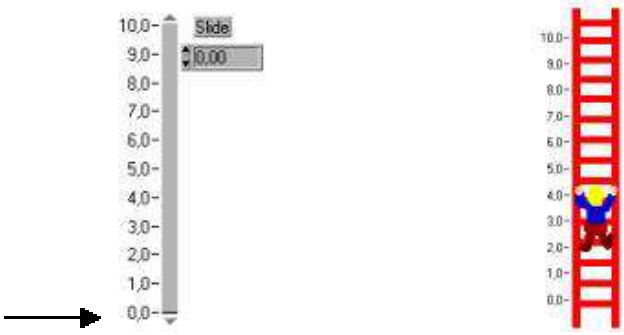
Kijelölünk a front panelről egy kontrollt ill. indikátort, mely működési tulajdonságai megfelelnek az új elemünk számára is, ezután az *Edit* menüből kiválasztjuk az *Costumize Control* menüpontot. Ekkor megjelenik a szerkesztő ablak (Control Editor) benne a kiválasztott objektummal. Minden elem több kisebb részegységből áll, melyek tulajdonságait (méret, szín, forma, stb.) külön-külön megváltoztathatjuk, a részeket áthelyezhetjük, ill. kitörölhetjük. Minden egyes rész új képét megrajzolhatjuk bármilyen rajzolóprogram segítségével. (Javasolt olyan rajzolóprogram használata, melynél a háttér átlátszó.) Ezután az elkészített rész kicserélhető az eredetivel. Ehhez a részegységre jobb egérgombbal kattintva a megjelenő pop-up menüből az *Import Picture* menüpontot kell választanunk.

Az így elkészített kontrollunkat elmenthetjük, a programunkban szereplő objektumunkat kicserélhetjük az új objektumra (*File* menü *Apply Changes* menüpontjával), ikont készítve az objektumunkhoz a *Controls* palettába is elhelyezhetjük azt.

A *11.1., 11.2, 11.3. ábrákon* példákat láthatunk az elemek átalakítására.

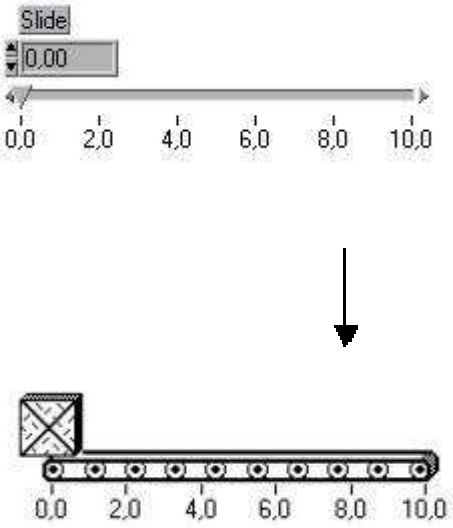
A LabVIEW által támogatott grafikus állományok:

CLP, EMF, WMF, GIF, BMP.



11.1. ábra. Csúszkából létra

11.2. ábra. Képcsoportból kártyacsoport



11.3. ábra. Csúszkából szállítószalag

11.1. Feladatok

1. Készítsünk kapcsolóból kétállapotú kézi működtetésű szelepet!

Megoldás:

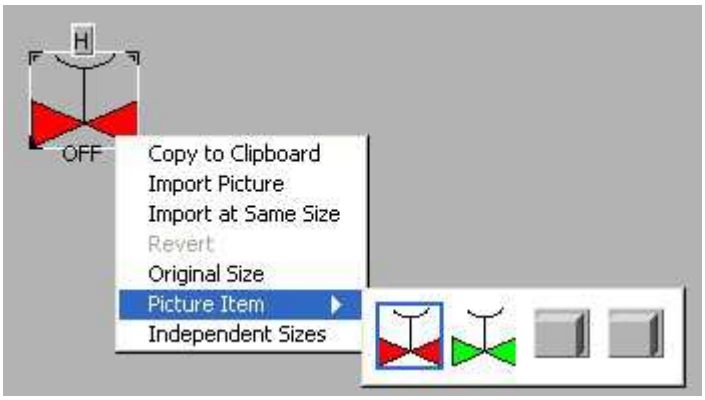
Először rajzoljuk meg a kéziszelepünk bekapcsolt, ill. kikapcsolt állapotbeli képét *(11.4. ábra)* olyan rajzolóprogrammal, mely a hátteret átlátszónak hagyja meg. Ez azért előnyös, mert az elem képe valószínűleg nem téglalap alakú lesz, és nem átlátszó háttér esetén kitakarhat dolgokat a felhasználói felületen, vagy csak nem lesz esztétikus.

Kikapcsolt állapot képe *Bekapcsolt állapot képe*

11.4. ábra. Szelep megrajzolt képe

Ha megvannak a rajzok, akkor helyezzünk el a felhasználói felületen egy hagyományos kapcsolót. Lehetőleg ne használjunk 3D-s elemet, mert az több alkotórészből áll (árnyék, aljzat, kapcsoló, felirat),míg a 2D-s elemnél csak a kapcsoló képe és a felirat látszik

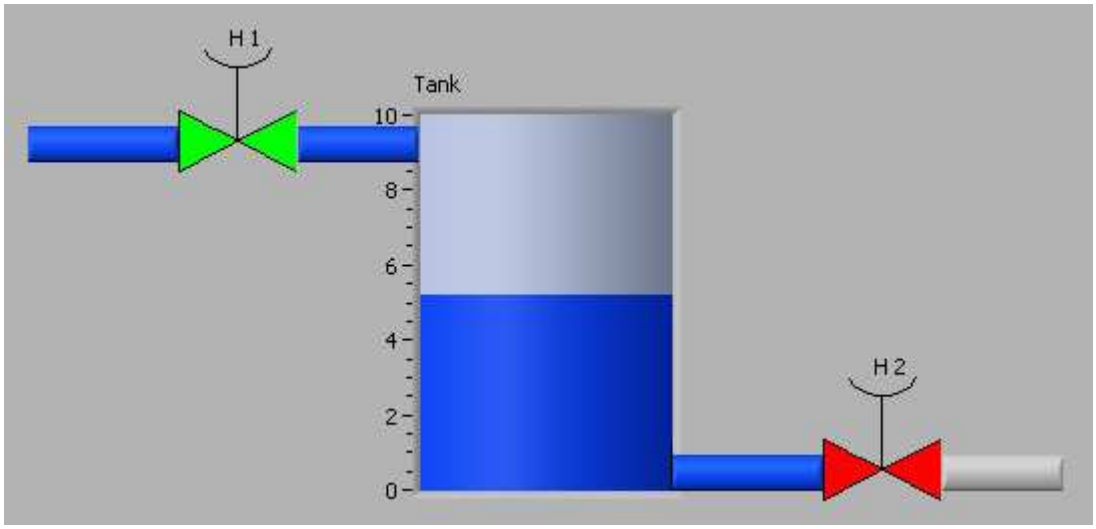
Jelöljük ki a kapcsolót, ezután az *Edit* menüből válasszuk ki az *Costumize Control* menüpontot. Ekkor megjelenik a szerkesztő ablak (*Control Editor*) benne a kiválasztott objektummal. Váltsunk át szerkesztő módból (*Edit Mode*) testreszabás módba (*Customize Mode*) (*Ctrl+M*), ekkor az elemünket részenként átméretezhetjük, mozgathatjuk, az egyes értékekhez tartozó képeket kicserélhetjük. Az Edit menüben válasszuk ki az *Import Picture from File* menüpontot, és töltsük be a kapcsoló igaz értékéhez tartozó képet. Ha ez megtörtént, az elemünk pop-up menüjében a *Picture Item* pontban válasszuk ki a kapcsolónk igaz képét *(11.5. ábra)*, majd kattintsunk az *Import Picture* vagy az Import at *Same Size* menüpontra. Ezzel meg is történt a kapcsoló igaz állapotában megjelenő kép cseréje.



11.5. ábra. Szelep képének beillesztése

Ugyanezt a műveletsort hajtsuk végre a kapcsoló hamis állására is. Váltsunk vissza szerkesztő módba (*Edit Mode*), és az *Operating tools* segítségével váltogatva a kapcsoló értékét kipróbálhatjuk annak működését.

A File menü *Apply Changes* menüpontjára kattintva az eredeti kapcsolónkat a felhasználói felületen kicserélhetjük az általunk készített új elemre *(11.6. ábra)*. Az új elemet el is menthetjük, ekkor CTL kiterjesztést kap, és a Controls paletta *Select a Control* pontjával bármely programunkba beilleszthetjük.

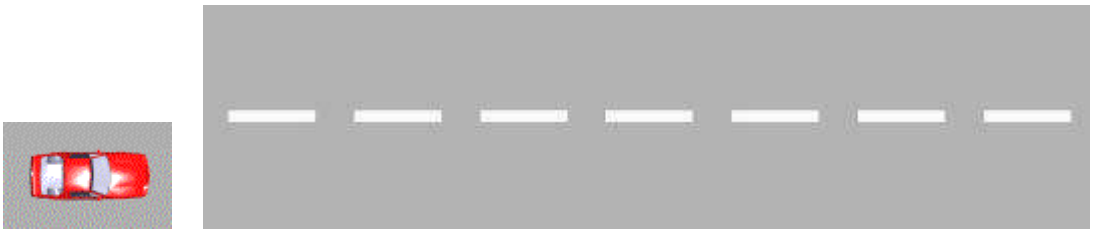


11.6. ábra. Saját elem használata a felhasználói felületen

2. Készítsünk programot, melyben egy csúszkából átalakított autó egy autóúton átsuhan a képernyőn!

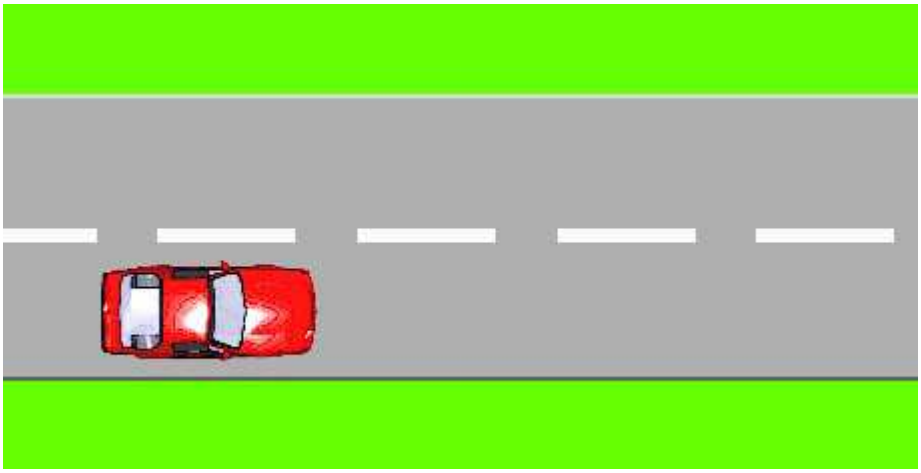
Megoldás:

Először rajzoljuk meg az autó és az út képét *(11.7. ábra)* olyan rajzolóprogrammal, mely a hátteret átlátszónak hagyja meg.



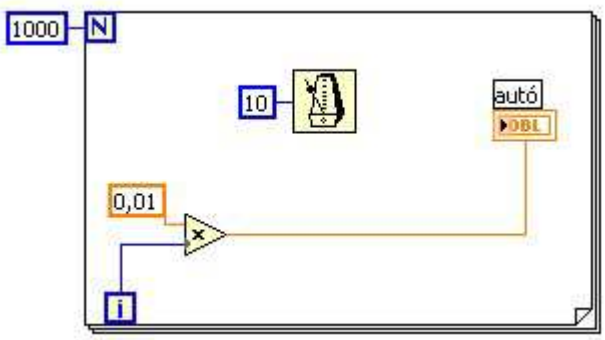
11.7. ábra. Autó és út megrajzolt képe

Ezután helyezzünk el a felhasználói felületen egy csúszkát, jelöljük ki, és az *Edit* menüből válasszuk ki az *Costumize Control* menüpontot. Ekkor megjelenik a szerkesztő ablak (*Control Editor*) benne a kiválasztott objektummal. Váltsunk át testreszabás módba (*Customize Mode*) (*Ctrl+M*). Cseréljük ki a csúszka hátterét az út képére, a Slide képét pedig az autóra. A File menü *Apply Changes* menüpontjára kattintva az eredeti elemet a felhasználói felületen cseréljük ki az általunk készített új elemre. Ezzel a Front panelt ki is alakítottuk *(11.8. ábra)*.



11.8. ábra. Az átalakított felhasználói felület

A program kódja igen egyszerű, mivel a csúszka értékét kell fokozatosan növelnünk egy cikluson belül, ill. helyezzünk időkésleltetést a programba, hogy szemünkkel követni tudjuk az autó mozgását *(11.9. ábra)*.



11.9. ábra. A program forráskódja

Miután összekötöttük a megfelelő elemeket, lépjünk át a Front panelre és teszteljük programunk működését.

12. Adatkapcsolati lehetőségek

A LabVIEW kifejlesztésének elsődleges célja a céleszközök helyettesítése volt, így nem meglepő, hogy általános mérőeszközként különféle bemeneti illesztéseket kell kezelnie.

12.1. Adatgyűjtő kártya

IBM PC/AT, EISA, IBM PS/2, Macintosh LC/LCII és SPARC sínekre csatlakoztathatjuk. A kártyán találhatók analóg, digitális és időzítő bemenetek ill. kimenetek. A LabVIEW kezelőprogramjai minden a National Instruments cég által gyártott eszközt kezelnek.

12.2. GPIB (General Purpose Interface Bus)

(Általános célú interfész sín)

A különálló eszközökkel való kommunikációt valósítja meg. Ilyen eszközök például a multiméter, az oszcilloszkóp, stb.. A kapcsolat megteremtésének legegyszerűbb módszere, ha egy GPIB kártyát helyezünk a számítógépbe, és ennek megfelelő installálása után ehhez csatlakoztatjuk műszereinket. A GPIB kapcsolatot kialakíthatjuk a soros porton keresztül is. A GPIB kapcsolat kezelőprogramjait a 488.2.VI és a GPIB.VI programok tartalmazzák.

12.3. Adatgyűjtés a soros porton keresztül

Korább népszerű adatátviteli módszer volt a számítógép és a perifériális eszközök között, mint például a nyomtató, a plotter és a programozható eszközök. Gyakran alkalmazzák például nagy távolságú adatkapcsolatoknál. A soros portot vezérlő alprogramokat a *Functions/Instrument I/O* menü *Serial* menüpontjában találhatjuk. A könnyebb kezelhetőség érdekében készültek el az úgynevezett VISA műveletek, melyek a soros port kezelésének állandó műveletsorait egy-egy alprogramban valósítják meg.

12.4. VXI busz használata

A VXI busz a műszerezési rendszerek platformja. A VXI kapcsolathoz egy adatgyűjtő keretet használunk, mely maximum 13 slot-ot tartalmaz. Ezekbe helyezhetők a moduláris felépítésű hordozókártyák. A kártyákra analóg, vagy digitális ki- és bemeneti modulok, ill. hőmérséklet modul kerülhet. A PLC és a számítógép közötti kapcsolatot a VME-AT készlet tartja fenn. A LabVIEW és a rendszer közötti kommunikáció subVI-okon keresztül történik. A VXI kapcsolatot kezelő alprogramok a VXI.VI állományban találhatók.

12.5. Feladatok

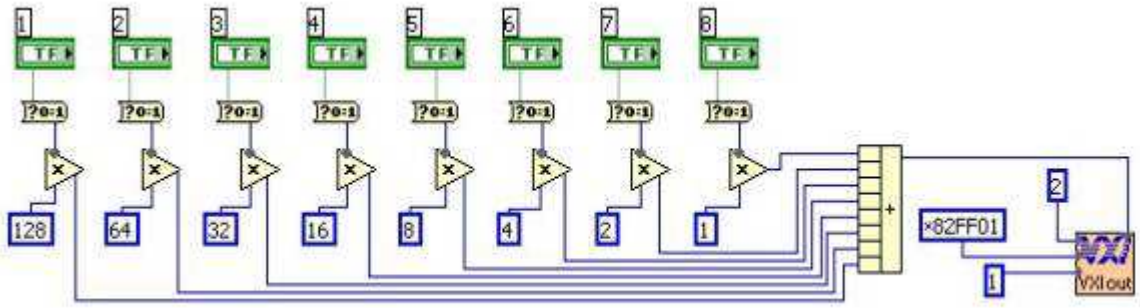
1. Készítsünk programot, melyben külső eszközzel kommunikálunk a VXI műveleteken keresztül! Olvassunk be az eszközről 8 bitnyi információt (1 bájtot alakítsunk át bitekké), majd írjunk ki 8 bitnyi információt!

Megoldás:

Az eszköz és a számítógép közötti kapcsolatot a VME-AT készlet tartja fenn. A LabVIEW és a rendszer közötti kommunikáció subVI-okon keresztül történik. A VXI.VI program tartalmazza azokat az alprogramokat, melyek szükségesek a VXI rendszerek LabVIEW-val történő kezeléséhez. Ezek közül a két legfontosabb a VXIin.VI, mely byte, word ill. longword típusú adatokat olvas be egy megadott VXI címről, és a VXIout.VI, mely byte, word ill. longword típusú adatokat ír ki egy megadott VXI címre.

Ezeket az adatokat célszerű felbontani (persze ezt az adott programozási feladat határozza meg). Ebből a célból készítsük el az *Eszk_IN.VI* és *Eszk_OUT.VI* programokat.

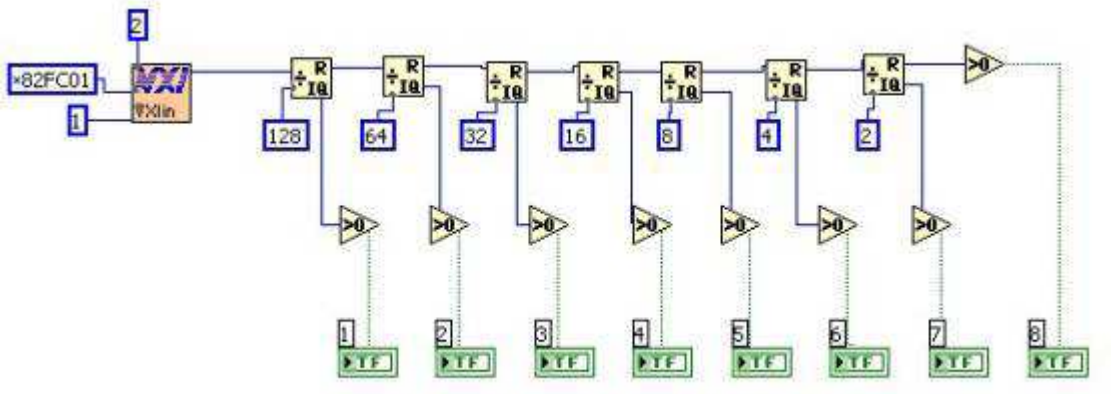
Az *Eszk_IN.VI* az eszköz első nyolc bemenetét kezeli *(12.1. ábra)*. A programnak nyolc darab logikai változót adunk át, melyekből egy byte típust képez. Ez a byte kerül az SM-DOUT modulra, amely továbbítja azt az eszköz felé. Ha a kérdéses bit értéke 1, akkor az eszköz hozzá tartozó bemenete aktív lesz, ha 0, akkor inaktív.



12.1. ábra. Az Eszk_IN.VI program forráskódja

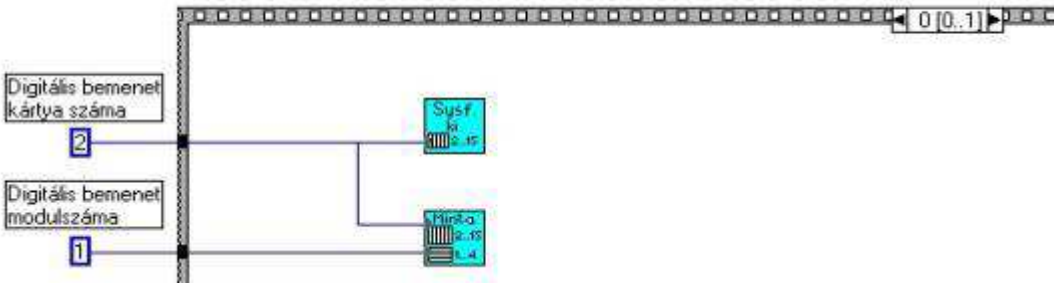
Az *Eszk_OUT.VI* az eszköz első nyolc kimenetét kezeli *(12.2. ábra)*. Az adat byte formájában érkezik meg a számítógépre az SM-DIN modulról, amit a programunk bontson bitekre. Ezekből a bitekből készítsünk logikai változókat, melyeket felhasználhatunk a további

programozásnál. A bitek az eszköz egyes kimeneteinek felelnek meg. Ha a bit értéke 1, az eszköz megfelelő kimenete aktív, ha 0, inaktív.



12.2. ábra. Az Eszk_OUT.VI program forráskódja

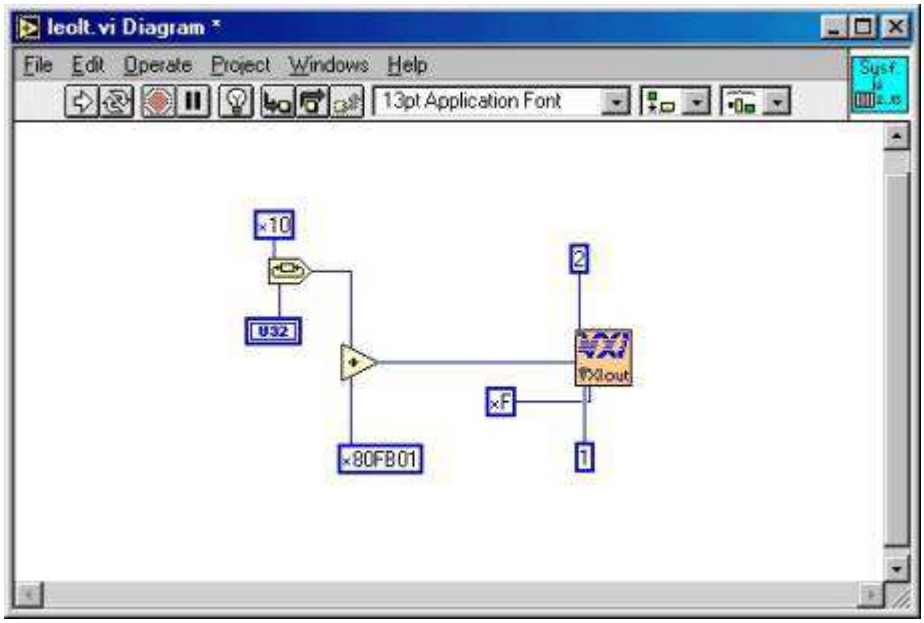
Ahhoz, hogy kommunikálni tudjunk az SM-DOUT és SM-DIN modulokkal, ki kell kapcsolni a megfelelő SMOD kártya *Sysfail LED*-jét, és el kell végezni az SM-DIN bemeneti modul pergésmentesítését *(12.3. ábra)*.



12.3. ábra. Kezdeti beállítások elvégzése

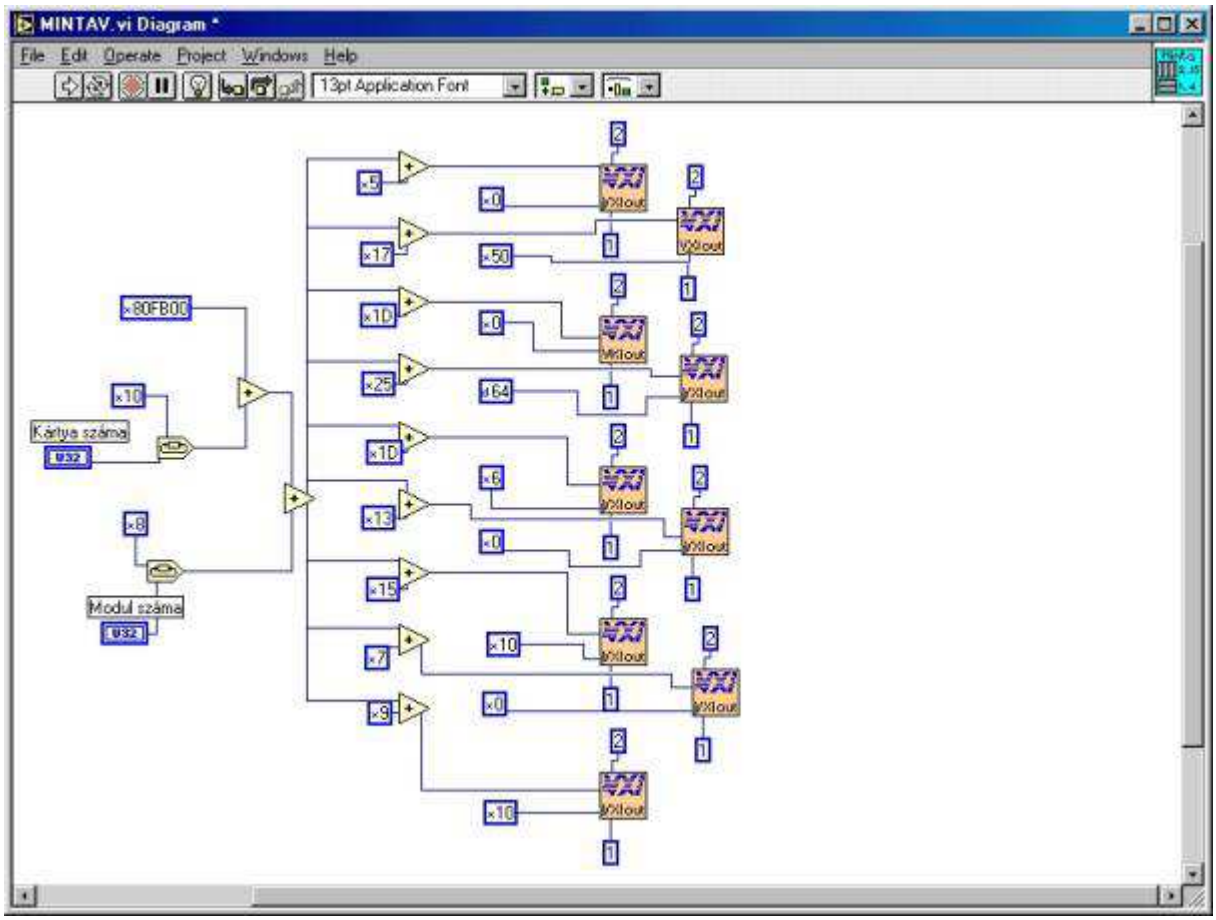
A programban meg kell adni a kártyaszámot, és a bemenetek modulszámát.

A Sysfail LED kioltását a *Leolt.VI* végzi *(12.4. ábra)*.



12.4. ábra. Az Leolt.VI program forráskódja

A pergésmentesítést a *Mintav.VI* végzi *(12.5. ábra).*



12.5. ábra. Az Mintav.VI program forráskódja

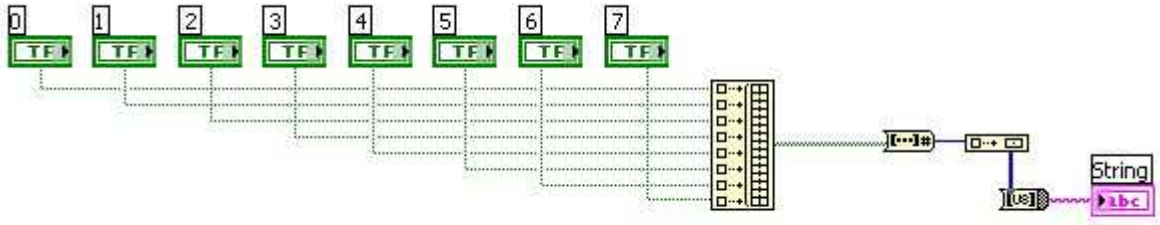
A program jellege nem engedi meg a párhuzamos lefutást, ezért szekvencia struktúrát kell alkalmaznunk. A szekvencia első lapjában végre kell hajtani a Sysfail LED kioltását és a pergésmentesítést, a másodikban olvassuk be az eszközről a 8 digitális jelet, a harmadikban írjuk ki a jeleket.

2. Készítsünk programot, melyben külső eszközzel kommunikálunk a soros porton keresztül! Olvassunk be az eszközről 8 bitnyi információt (1 bájtot alakítsunk át digitális jelekké), majd írjunk ki 8 bitnyi információt!

Megoldás:

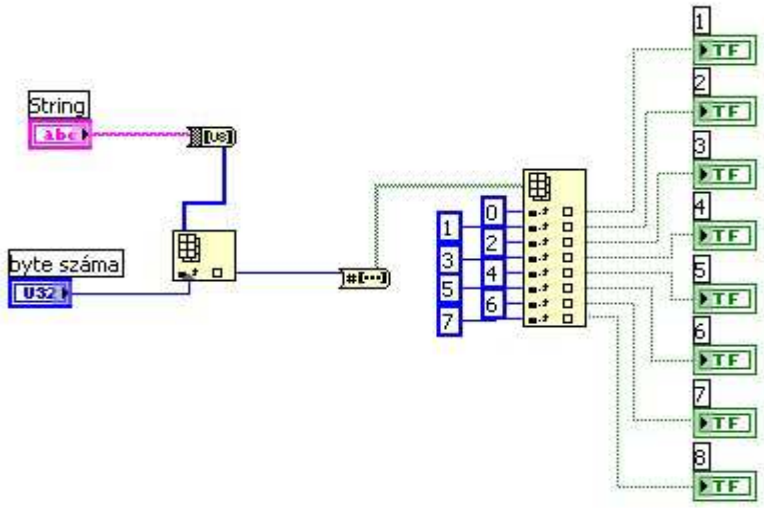
A soros kommunikáció kialakítására is készítsünk alprogramokat, hasonlóan a VXI kommunikációhoz. Ez az adatkapcsolat azonban jellegében is eltér az előzőtől. Míg abban az esetben biteket továbbítottunk az eszköz felé, itt stringet kell a soros portra küldeni, ill. onnan kiolvasni. Ezért készítsük el a konverziós alprogramokat.

A *8bit-böl_string.VI* 8 bitből 1 byte-nyi stringet képez *(12.6. ábra)*.



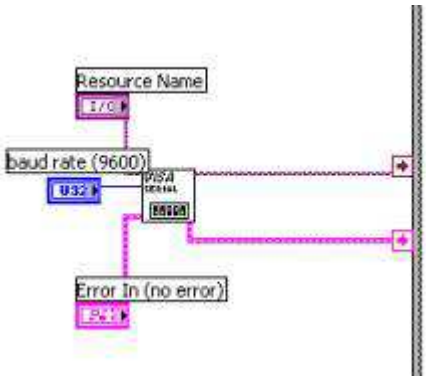
12.6. ábra. Az 8bit-böl_string.VI program forráskódja

A *string-böl_8bit.VI* egy string valahányadik byte-ját 8 bitté konvertálja *(12.7. ábra)*.



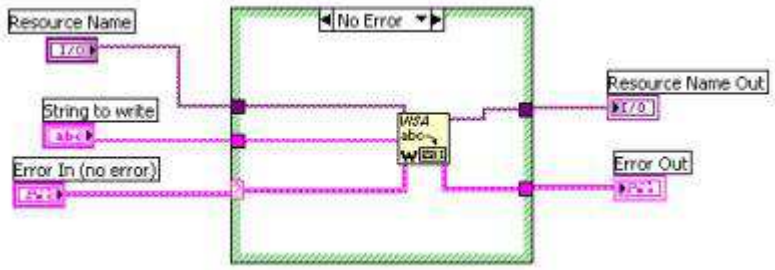
12.7. ábra. Az string-böl_8bit.VI program forráskódja

A soros kommunikációban a műveleti egységeknek megszabott sorrendjük van, ezért egy szekvenciát alkalmazva kell elválasztanunk egymástól ezeket. A kommunikáció megkezdése előtt inicializálni kell a portot, ez látható minden program első (0.) szekvenciájában *(12.8. ábra)*.



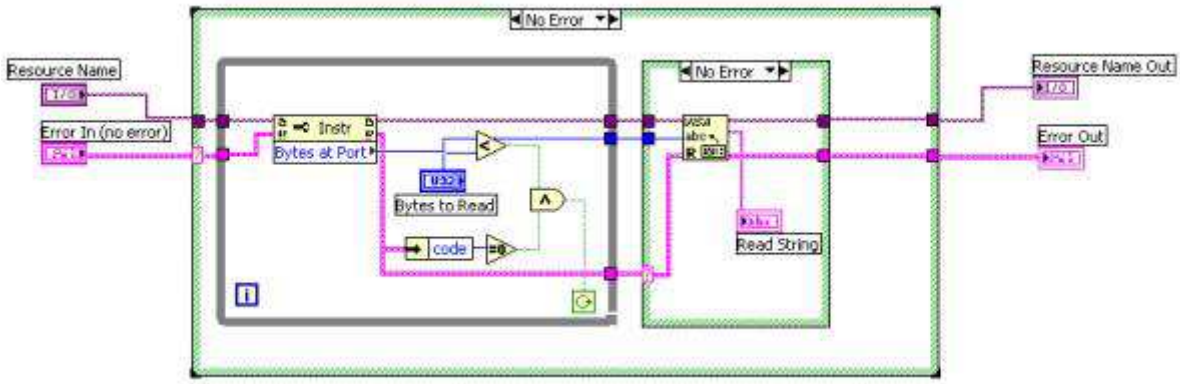
12.8. ábra. Soros port inicializálása

Majd következhet a portra való írás (*Soros_ki.VI*) *(12.9. ábra)*. Hiba esetén nincs írás.



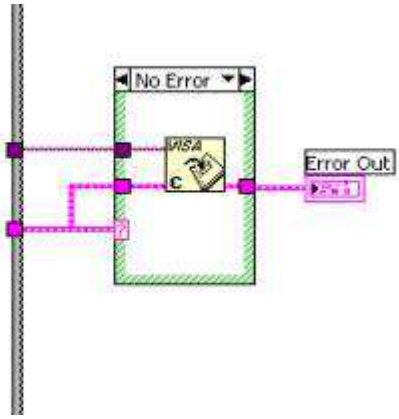
12.9. ábra. Az Soros_ki.VI program forráskódja

Azután a port kiolvasása (*Soros_be.VI*) *(12.10. ábra)*, ahol a program vizsgálja a portot, és akkor történik a beolvasás, ha a kellő számú bájt, jelen esetben 8, összegyűlt. Hiba esetén nincs beolvasás.



12.10. ábra. Az Soros_be.VI program forráskódja

És a legvégén a kommunikáció lezárása *(12.11. ábra)*.



12.11. ábra. A kommunikáció lezárásának kódja

Mindez akkor jöhet létre, ha az inicializálás során nem történt hiba (ezt jelzi a case struktúra zöld, no error kerete).

A program részeit szekvenciálisan kell futtatni. A szekvencia első lapján végezzük el az inicializálást, a második lapon olvassunk be egy bájtnyi információt, és ezt alakítsuk bitekké, a harmadik lapon a kiírandó biteket konvertáljuk sztringgé, és végezzük el a kiírást, majd a negyedik lapon zárjuk le a kommunikációs portot.

12.6. Ellenőrző kérdések

1. Mi szükséges ahhoz, hogy egy programot alprogramként használhassunk?

2. Mi a szerepe a *Connector*-nak?

3. Hogyan történik az általunk elkészített SubVI beillesztése egy másik programba?

4. Mi a lényege a saját elem készítésének?

5. Tudunk-e létrehozni alapvetően új működésű saját elemet?

6. Milyen külső adatkapcsolati lehetőségekkel dolgozhatunk a LabVIEW-ban?

7. A Soros port kezelésének milyen lépéseit kell végrehajtanunk a kommunikáció során?

Irodalomjegyzék

1. National Instruments Corporation: *Getting started with LabVIEW*, 2000

2. National Instruments Corporation: *LabVIEW Tutorial*, 2000

3. National Instruments Corporation: *LabVIEW User Manual*, 2000

4. National Instruments Corporation: *Development Guidelines*, 2000

5. National Instruments Corporation: *Instrument I/O VI Reference Manual*, 2000

6. Jeffrey Travis: *LabVIEW for Everyone*, 2002

7. J. Conway, S. Watts: *A Software Engineering Approach to LabVIEW*, 2003

A National Instruments Corporation weboldalán további hasznos információkat olvashatunk (leírás, tutorial, bemutató anyag) és tölthetünk le (demo verzió, kiegészítések) a LabVIEW-val kapcsolatban (http://www.ni.com/labview/).