DonGó Béta

Programozható vezérlőpanel programozási, informatikai és elektronikai alkotásokhoz

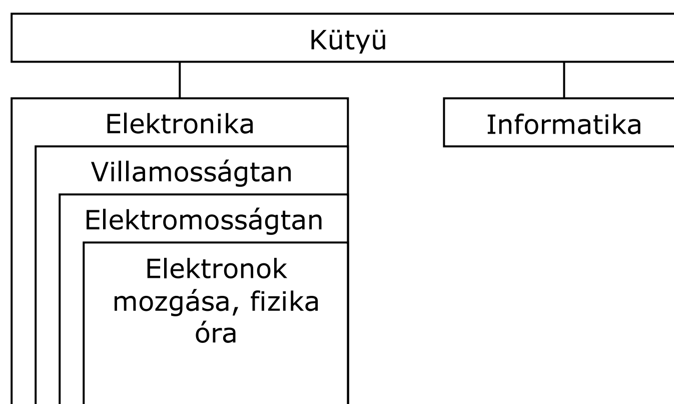
DonGó – Az elektronika mindenkié ... 3
Elektronok: legutoljára a fizika órán ... 3
Villamosságtan: kapcsolók, drótok, lámpák ... 3
Elektronika: okos alkatrészek, integrált áramkörök ... 3
Kütyü: programozott kicsi számítógépek ... 4
Programozni! ... 4
Számítógép – vagy nem? ... 5
Mit programozzunk? ... 6
Próbáld ki ... 7
Bekapcs! ... 7
Villanj! ... 8
Villogó led ... 9
Szép program ... 10
Várjunk egy gombra ... 12
Feltételek, feltételek ... 13
Összetett feltételek ... 15
Konstansok ... 16
Deklaráció ... 16
Változók ... 17
Szöveges változók ... 18
Számoljunk a változókkal! ... 19
A for-ciklus ... 20
Szakácskönyv ... 21
Pascal referencia ... 23
DonGo referencia ... 26
Port kezelés (be-kimenetek) ... 26
Hang kezelés ... 26
Beszéd kezelés ... 27
Timer (időzítő) kezelés ... 28
Kiírás ... 28

DonGó – Az elektronika mindenkié

Emlékszel a fizikaórák álmos reggeleire, amikor Amper, meg Volt, meg egyéb riadalmakkal próbálták megmutatni Neked, hogy mi is az az elektromosság? Mi mindent tudsz róla? Vajon tudsz-e zenélő bilit készíteni? Ugye még nem? Akkor jó helyen jársz.

Az elektromosság a fizikaórán arról a jelenségről szól, *ami* a telefonodat, számítógépedet és még sok más dolgot is működtet. Azt, hogy *hogyan* működteti ezeket, már lehet, hogy nem ismered. Vajon *hogyan* építhetsz Te is ilyen elektromos dolgokat? Most megmutatom. Nem fogok elektronokról, Amperekről beszélni, mert nincs szükséged ezekre. (Meglepőt mondok: A péknek sincs szüksége a búzatermesztés részleteire ahhoz, hogy tökéletes kenyeret süssön.)

Az elektronika behálózza mindennapjainkat, és alapot teremt az informatika számára. Sok esetben az informatika és az elektronika összefonódik: így készülnek mindennapjaink okos készülékei, mint például az önmagát beállító kvarcóra, vagy a mobil telefon. Én így képzelem el a világot:



Elektronok: legutoljára a fizika órán

Eddig vagy tanultál elektromosságtant (és ott az elektronok mozgását), vagy nem. Ha nem tanultál még ilyet, az sem baj, ne aggódj, most sem foglak ezzel fárasztani.

Villamosságtan: kapcsolók, drótok, lámpák

A villamosságtan arról szól, hogyan tudjuk az elektromosság jelenségét hasznunkra fordítani. Hogyan szerzünk villanyt (például elemből), hogyan szállítjuk a villanyt (például dróton) és mit csinálunk a villannyal (például egy lámpába vezetjük, ami világít). Ez nagyon hasonlít a villanyszerelők munkájához: kapcsolókat, lámpákat, konnektorokat kötnek össze vezetékekkel. A régebbi autók elektromos rendszere sem volt ettől bonyolultabb: kapcsolók, és lámpák. A DonGó Zéró készlet segítségével Te is beszállhatsz ebbe.

Elektronika: okos alkatrészek, integrált áramkörök

Az elektronika arról szól, hogy okos elektronikus alkatrészek összekapcsolásával okosan oldjunk meg problémákat. Például, egy sötétben magától felkapcsolódó lámpa már az elektronika kategóriájának az aljába tartozik. Az

elektronika kategória teteje a mobiltelefonok, számítógépek elektronikája. Az elektronikus eszközök építőelemei az integrált áramkörök (IC-k). Szinte minden funkcióra van egy-egy ilyen IC, ezeket mint a legó darabokat egymáshoz lehet kötni. A DonGó Alpha készlet segítségével Te is eljuthatsz erre a szintre.

Kütyü: programozott kicsi számítógépek

Manapság szinte minden dologban van valamennyi elektronika és valamennyi informatika. A mobiltelefon, az MP3 lejátszó, de már a legtöbb mikrosütő is az elektronika és informatika keverékéből épül fel. Lehet, hogy most meglepődsz, de a legtöbb kütyüben van egy picike számítógép, ami a kütyü működését vezérli. Például, a tévé távirányítójában is egy pici számítógép van, amit mikrovezérlőnek, vagy mikrokontrollernek neveznek. Nem azért van benne számítógép, mert sokat kellene számolnia, hanem azért, mert így a legegyszerűbb, legolcsóbb a kütyüt elkészíteni. A kütyük világa (amit angolul embedded systems-nek hívnak) elsősorban ezekből a pici számítógépekből építkezik. A DonGó Beta készlettel Te is készíthetsz összetett kütyüket: órát, beszélő ajtót, és még sok egyebet.

Programozni!

Elektronikus kütyüket lehet csinálni gombokból, ledekből, logikai kapukból és egyéb építőelemekből. A 80-as évektől könnyen elérhetővé váltak a mikrovezérlők – olyan integrált áramkörök, amik egy teljes számítógépet tartalmaznak. Ezen pici számítógép programja határozza meg, hogy mi történjen a kütyüvel, hogyan reagáljon a kütyü a bemenetei változására.

Programot csinálni...

- ***könnyű:*** sokkal könnyebb, mintha többszáz alkatrészt kellene fejben tartani.
- ***biztonságos:*** nem valószínű, hogy bárki megégeti a mancsát, vagy tönkretesz valami alkatrészt.
- ***gyors:*** sokkal gyorsabb, mint bármi más megoldást választani a problémára
- ***rugalmas:*** ha valamit meg kell változtatni, elég csak a programot átírni
- ***egyszerű:*** egyre könnyebben használható programnyelvek és programozási eszközök vannak

Ezek épp elég okot adnak rá, hogy manapság minden kütyüben ami picit is bonyolultabb a vízforralónál, legyen egy programozható kis számítógép. Ilyenek vannak már 20 éve a távirányítókban, a mikrosütőkben, a mobiltelefonokban, a zenelejátszókban, játékautomatákban, riasztóban, de sokszor előfordulnak akár egy tűzhelyben is.

Számítógép – vagy nem?

A laptopod és a 200Ft-os mikrovezérlők között csak mennyiségi különbség van. A laptopokban és asztali számítógépekben sok memória van, a winchesteren sok program és adat fér el. A gép „bemente” a billentyűzet és az egér, a „kimenete” meg a monitor, esetleg hangszórók, nyomtató.

Egy mikrovezérlős játékautomatában kevés memória van, egyetlen fix programot futtat, a bemenete a gombok és a pénzelfogadó, a kimenete a hengereket forgató motorok, és vagy 200 színes csillogós lámpa, valamint egy aprópénz-kihajigáló (hopper) nevű cucc.

Egy távvezérlő bemenetei a gombok, a kimenete egyetlen infravörös led, ami a megnyomott gombnak megfelelő rövid és hosszú villanásokat produkál.

Mit programozzunk?

A mikrovezérlős dolgokban nagy könnyebbség, hogy ezek egyetlen dologra készültek, és egyetlen program van bennük. Ismerjük pontosan, hogy milyen eszközöket dugtunk a mikrokontrollerre, és ezek az eszközök általában olyan egyszerűek, hogy kezelésük gyerekjáték.

Az asztali számítógép esetében manapság egy kicsit bonyolultabb a helyzet. Először is, ezek általános célú számítógépek, sok programmal. A mi programunknak szépen kell viselkednie, és nem szabad más programok működését akadályoznia. A gépre nagyon trükkös és bonyolult eszközöket (pld. videokamerát) is csatlakoztathatunk, ráadásul nagyon sok gyártó nagyon sokféle termékét – amiket természetesen más-más módon kell kezelni. Nehéznek tűnik? Ha nehéz lenne, senki nem csinálna programokat, és nem fejlődne töretlenül ez az iparág ☺ Nyilván nem nehéz, mert segítségünkre van egy csoda-program, az operációs rendszer.

Az operációs rendszer:

- Elosztja az erőforrásokat (pld. minden program a saját ablakába rajzolhat, egyszerre több program futhat (osztozhat) a memórián és a processzoron)
- Kezeli és elfedi a hardverkülönbségeket (pld. minden videokamerának saját kezelőprogramocskája (driver) van, ezek áthidalják a különbségeket, hogy a Te programod bármelyik kamerát egyformán tudja kezelni)
- Megvalósít gyakran haszált funkciókat (pld. file-ok kezelése a winchesteren/usb drive-on/hálózaton át)
- Jogosultság- és hozzáférés-ellenőrzést végez (pld. egy egyszerű felhasználó nem törölheti le az operációs rendszer programjait, vagy mások adatait)

A mikrokontrolleres esetben egyetlen program – a mi programunk létezik, ezért nincs mit elosztogatni az egyszerre futó programok között. Nincsenek hardverkülönbségek sem – hiszen rajtunk múlik, milyen hardvert használunk. Egyszerű a hardver, annyira, hogy drivert sem kell írni. Nincsen „mások adata” hiszen nincs több felhasználó (sőt, van, amikor egyetlen „ember” felhasználó sincs, pld. a fűtésszabályzó rendszerek a hőmérsékletre kíváncsiak, nem emberekre.)

A fentiek miatt az asztali számítógép esete manapság egy picit bonyolultabb, mert több háttérismeretet igényel. Ezért először mikrovezérlőt programozunk, majd utána megtanuljuk, hogyan lehet asztali gépet programozni. (Érdekes megjegyezni, hogy a 80-as években az asztali számítógépeken sem volt komoly operációs rendszer, egyszerre csak egy programot futtattak, és egy mai mikrovezérlő simán lenyomja a legtöbb ilyen gépet teljesítményben. Nem megmondtam: ezek bizony egyformák.)

Próbáld ki

A DonGo Beta egy mikrovezérlős panel, 12 be-kimenettel, és egy hang-kimenttel. Ezt lehet többek között Pascal nyelven is programozni.

Ennek a vezérlő panelnek működését szimulálja az a szimulátor program, amivel bármely asztali számítógépen tudunk programot írni. Az így megírt program fut, működik, és mindent úgy csinál, mintha a tényleges kütyün futna.

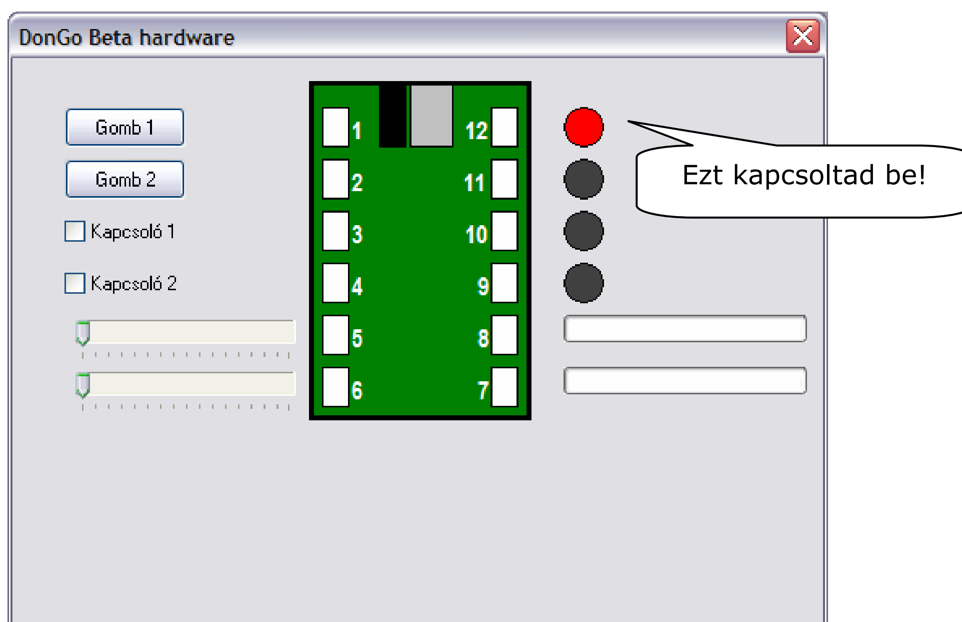
Ha a géphez csatlakoztatod a DonGó Béta panelt, akkor valóságosan működő kütyüt kapsz. A DonGó Béta panel 12 csatlakozójára a DonGó Alfa készlet érzékelőit, vezérlőit, illetve feldolgozómoduljait csatlakoztathatod.

Bekapcs!

Indítsd el a szimulátor programot! Egy szövegszerkesztő ablakot látsz – ide kell beírni a programot, hogy mit is csináljon a kütyünk. Írd bele a következő programot:

```
begin
      setPort(12,1);
end.
```

Nyomd meg a zöld háromszög gombot (vagy az F9 gombot, vagy válaszd ki a Run menüből a Run menüpontot), és a program elindul. Egy külön ablakban, a szimulátor képernyőjén látod a DonGo Beta panelt, és a 12-es sorszámú csatlakozója melletti piros ledet. Ezt most Te kapcsoltad be!



A programot Pascal nyelven írtuk. Sok programnyelv létezik a világon, de az emberek azokat szeretik jobban, amit maguk is el tudnak olvasni. A Pascal egy ilyen nyelv: minden utasítás egyszerű angol szó, értelmes jelentéssel.

A program elején a **begin** azt jelenti, hogy elkezdeni, a végén az **end** meg azt, hogy befejezni. A program közepe pedig egyetlenegy ***utasítás***: **setPort**. A set azt jelenti, beállítani, a port pedig azt jelenti, kikötő... (Ebbe a kikötőbe nem hajók jönnek-mennek, hanem be- és kimenő információk. A port jelentése tehát a DonGo panelünk csatlakozási pontja. (Pld.: a számítógéped USB portja = USB csatlakozási pont.))

A **setPort** mellé ***zárójel***be írtunk két mágikus számot. A 12 (ugye már kitaláltad) azt jelenti, hogy melyik portra vonatkozzon az utasítás. A második szám, az 1 pedig azt jelenti, hogy kapcsoljuk be (a 0 pedig, hogy kapcsoljuk ki). Ez a két érték az utasítás ***paramétere*** . (A paraméter nem a félelem mértékegysége.) A paraméterektől függően picit más-más dolgot csinál az utasítás. Például, a **setPort(12,0)** kikapcsolja a 12-es portot, a **setPort(11,1)** pedig a 11-es portot kapcsolja be.

Az utasítások végét ***pontosvessző***vel zárjuk. (Ez amolyan szokásos dolog a legtöbb programnyelven. A Pascalban vannak érdekes fura kivételek: a **begin** és az **end** az nem „utasítás" (hiszen semmit nem csinál ténylegesen), emiatt ezek után nem kell pontosvesszőt tenni. Az **end** után viszont sokszor látsz majd pontosvesszőt a példákban. Ez a pontosvessző nem az **end**, hanem az az *előtti* utasítás végét jelzi. A pontosvessző-mizériára vissza fogunk térni, nagyon logikus, ha egyszer megérti az ember.

A program végét ***pont*** zárja. (Ennek sincsen semmi logikus magyarázata, csak igyekeztek a Pascalt hasonlóvá tenni az emberi nyelvekhez. Ott is pont van a mondat végén.)

Villanj!

Próbáld ki a következő programot:

```
begin
      setPort(12,1);
      delay(500);
      setPort(12,0);
      delay(500);
      setPort(12,1);
end.
```

Ez a program a már megismert **setPort**-ot használja, és a **delay** utasítást. A delay jelentése késleltetés, a **delay(500)** az 500 milliszekundum (ezredmásodperc) (azaz fél másodperc) ideig nem csinál semmit.

A programba több utasítást írtunk egymás alá, ezeket sorban hajtja végre a számítógép. Először bekapcsolja a 12-es porton lévő ledet, aztán kicsit vár, aztán kikapcsolja, aztán kicsit vár, végül ismét bekapcsolja. Indítsd el a progit! Azt látod, hogy a 12-esen lévő led bekapcsol, kikapcsol, bekapcsol – pont, ahogy megírtad.

Villogó led

Jó lenne, ha sokáig villogna ez a led. De csak nem akarjuk ezerszer leírni, hogy **`setPort(12,1)`** meg **`setPort(12,0)`**. Nyilván van erre okosabb megoldás: ez pedig az, ha megkérjük a gépet, ismételgessen meg egy pár utasítást.

Próbáld ki ezt:

```
begin
      while 1=1 do
      begin
            setPort(12,1);
            delay(500);
            setPort(12,0);
            delay(500);
      end;
end.
```

Ha elindítod, a gép ismételgeti a setPort-jainkat, és a led villogni fog, igaziból! Annyira szorgosan ismétli a gép az utasításokat, hogy abba se hagyha, csak ha megnyomod a piros négyzet (stop) gombot a DonGo ablakán (vagy a Ctrl+F2-vel is meg tudod állítani a programot).

Az utasítások ismételgetését ***ciklus***nak nevezik. Az utasítások végtelen ismételgetését meg végtelen ciklusnak – ezt csináltuk most, hiszen ha nem állítod le a programot, végtelen sokáig megy a ciklus, és villogtatja a ledet.

Többféle utasítás van ciklusok készítésére, a **`while`** az egyik legkedvesebb. Nem kis fantáziával az ezzel készült ciklusokat *while-ciklusnak* nevezik. A while jelentése *amíg*. A while után van egy összehasonlító ***kifejezés***. Amíg a kifejezés értéke igaz, a ciklus belsejében lévő utasításokat ismételgeti. Jelen esetben azt vizsgáljuk, vajon 1 egyenlő-e 1-el. Nos, a válasz az, hogy ez igen, igaz. Sőt, ez örökké igaz, így a ciklusunk is örökké fut. Az összehasonlító kifejezés bármilyen relációs művelet lehet (kisebb: <, nagyobb: >, egyenlő: =, nem-egyenlő: <>, kisebb vagy egyenlő: <=, nagyobb vagy egyenlő >=) és a kifejezésbe nem csak fix számokat, hanem bármi számot visszaadó műveletet is írhatunk. (Egyelőre még nem tudunk ilyen műveleteket, de a matematikai műveletekkel lehet vicceskedni: **`while 1+3=4 do`**...)

A ciklus belseje – amit ismételget – az a **`do`** után következik, és ***ciklusmag***nak nevezik. A ciklusmag után pontosvessző zárja a **`while`** utasítást. (Dehát ott van még ezer pontosvessző! Mindjárt kimagyarázom.) A ciklusmag lehet egyetlenegy ***elemi utasítás***, pld:

```
while 1=1 do delay(500);            while feltétel do utasítás;
```

Vagy a ciklusmag lehet begin...end közé zárt több utasítás, azaz ***utasításblokk***:

```
while 1=1 do                        while feltétel do
begin                               utasításblokk;
      delay(500);
end;
```

A while pontosvesszője a végén van (ha a végén end van, akkor az end végén), ugye tényleg, és nem csaltam?

Tudjuk, hogy a while a do után következő utasítást vagy blokkot ismételgeti. Mit csinál tehát a következő progi?

```
begin
      setPort(11,1);

      while 1=1 do
      begin
            setPort(12,1);
            delay(500);
            setPort(12,0);
            delay(500);
      end;

      setPort(11,0);
end.
```

Nos, először bekapcsolja a 11-es portra rakott ledet. Utána jön a villogó rész... aminek ugye sosincs vége, ezért a legutolsó utasítás, a 11-es port kikapcsolása *soha nem fog végrehajtódni.* Logikus?

Szép program

A progit szépen illik írni. Nem a számítógép kedvéért – az mindent megért. Sokkal inkább magunk kedvéért: hiszen egy hosszabb programot sokszor fogunk átolvasni, elolvasni, és ilyenkor kényelmes, ha jól áttekinthetően néz ki.

A Pascalban kis- és nagybetűket egyaránt használhatsz az utasításokban: SETPORT, setport, setPORT, SETport, SeTpOrT mind-mind ugyanazt jelenti. Sok-sok okos ember vacakolt azzal, hogy alkosson egy olyan szabályrendszert, ami hosszú távon is könnyen olvashatóvá teszi a programokat. Ezek pediglen:

- Minden dolgot kisbetűvel írunk, mert azt könnyebb elolvasni. (pld. **`while`**) Ha az utasítás több szóból álló szókapcsolat, akkor az olvasás könnyítésére a kapcsolódó szavak első betűjét nagybetűvel írjuk. (pld. **`setPort`**, vagy **`kukacKesztyuUjj`**). (Ezt a váltakozó kisNagyBetűs írást nevezik angolul camelCase-nek, mert „hullámzik" mint a teve (*camel*) háta.)
- Egy sorba egyetlen utasítást teszünk. (Bár lehetne írni, hogy **`setPort(12,1);delay(500);setPort(12,0);delay(500);`** de ha ezt egyszer elkezdjük, akkor nagyon hosszú programsoraink lesznek.) A program a „hosszában" azaz a sorok száma mentén növekszik, a „szélességében" nagyon nem illik növekednie. Ha „széles" a program, akkor mindíg görgetni kell még vízszintesen is, hogy végig lehessen olvasni – ami kényelmetlen.)
- Az utasításokat bekezdésekkel tagoljuk. A bekezdés lehet 4 szóköz, vagy egy TAB (erre való a TAB gomb). A bekezdéseknek az az értelme, hogy lássuk, mely utasítások melyik utasításblokkhoz tartoznak. Az ökölszabály: minden begin és end új sorba kerül, és a begin..end közti utasításokat egy bekezdéssel beljebb tesszük. A bekezdéselést nevezik ***indent***álásnak.

- Nyugodtan lehet üres sort tenni a programba, így elválasztva a különálló részeket. Hasonlóan lehet megjegyzéseket (***komment***eket) tenni a programba a // jellel kezdve. A kommenteket figyelmen kívül hagyja a számítógép – viszont nekünk segít megérteni, mit is csinál a program. Kommentek feleslegesek a nyilvánvaló utasításokhoz: felesleges komment: **`// bekapcsoljuk a 12-es portot.`** A kommentek pratkikusak a bonyolultabb, nem-nyilvánvaló dolgokhoz: **`// most villogtatjuk a ledet a 12-es porton.`**

Az alábbi két példa ugyanazt a programot tartalmazza. Vajon melyikről könnyebb kitalálni, mit csinál?

```
begin setport(11,1); while 1=1
do BEGIN
setPort(12,1);
 delay(500);    setPort(12,0);
DELAY  ( 500);
      end;end.
```

```
begin
      setPort(11,1);

      //--villogtatjuk a 12-est
      while 1=1 do
      begin
            setPort(12,1);
            delay(500);
            setPort(12,0);
            delay(500);
      end;
end.
```

```
begin
      setPort(11,1);

      //--villogtatjuk..
      while 1=1 do
      begin
            setPort(12,1);
            delay(500);
            setPort(12,0);
            delay(500);
      end;
end.
```

A zöld a programunk magja, a sárga a while ciklus magja.
A begin-end-ek mindíg párban vannak, mindíg egymás alatt, így jól látszik, hogy milyen utasítások vannak közöttük.

Várjunk egy gombra

Csináljunk olyat, hogy akkor kezdjük a villogást, ha a legelső gombot megnyomták. Próbáld ki ezt:

```
begin
      while getPort(1)=0 do ;

      while 1=1 do
      begin
            setPort(12,1);
            delay(500);
            setPort(12,0);
            delay(500);
      end;
end.
```

Ahogy már tudjuk, a while ciklus egy ***összehasonlító kifejezés*** – másnéven ***feltétel*** – értékét ellenőrzi. Az első while ciklusban egy mágikus `getPort(1)`-et írtunk. Ez megnézi az 1-es portra tett gomb állapotát, és az értéke 0 lesz, ha a gombot nem nyomták meg, és 1 lesz, ha a gomb meg van nyomva.

Ha a getPort értéke 0, akkor a feltétel teljesül, és a while ciklus végrehajtja a ciklusmagját – ami – hmm, hol a ciklusmag? A do és a pontosvessző közt semmi sincs. Üres a ciklusmag, hiszen pontosan azt szeretnénk, hogy semmit se csináljon, amíg a gombot nem nyomták meg.

Mihelyst a gombot megnyomják, a getPort értéke 1 lesz, a while feltétele nem teljesül, és megy tovább a program végrehajtása a szokásos ledvillogtató részre.

Feltételek, feltételek

Csináljunk olyat, hogy ha megnyomják az egyik gombot, akkor világítson a 12-es porton a led. Ha elengedik, kapcsolódjon ki a led.

```
begin
      while 1=1 do
      begin
            if getPort(1)=1 then
                  setPort(12,1)
            else
                  setPort(12,0);
      end;
end.
```

Huhh? Nem kell megijedni, az egyetlen újdonság az **`if-then-else`** szerkezet, magyarul ha-akkor-egyébként. Az if után írunk egy ***összehasonlító kifejezés*** – másnéven ***feltétel*** t– ami ha igaz, akkor a then utáni részt hajtja végre a gép. (Hogy mi is az, hogy igaz? A while ciklus résznél tudod elolvasni.) Ha a feltétel nem igaz, akkor az else utáni részt hajtja végre. Vagy az egyik, vagy a másik részt hajtja végre a gép – mintha „elágazáshoz" érkezne a program az útján, ezért az if-then-else-t ***elágazás***nak is nevezik.

A programunk a következőt csinálja: ha a getPort 1-et ad vissza (mert meg van nyomva a gomb), akkor a 12-es porton bekapcsolja a ledet, egyébként meg kikapcsolja a 12-es porton a ledet. Ezt az egész dolgot pediglen folyamatosan ismétli – hiszen ezt az egészet egy végtelen while ciklusba tettük.

A dologban az a felettébb praktikus, hogy itt nem várunk semmire, soha, és a külső while ciklus miatt a gép ezerrel nézegeti a getPort értékét állandóan. Mit kellene tenni, ha mondjuk a 2-es gombot és a 11-es ledet is szeretnénk „összekötni", de most éppen fordítva? Ha a 2-es gomb nincs megnyomva, akkor világítson a 11-es led, különben pedig nem. (Ez pont a hűtőszekrény-nyomógomb működés.)

```
begin
      while 1=1 do
      begin
            //-- Ha az 1-es gombot nyomják
            //-- világít a 12-es led
            if getPort(1)=1 then
                  setPort(12,1)
            else
                  setPort(12,0);

            //-- Ha a 2-es gombot elengedik
            //-- világít a 11-es led
            if getPort(2)=0 then
                  setPort(11,1)
            else
                  setPort(11,0);
      end;
end.
```

Az if-then-else is olyan, mint a while: vagy egyetlen utasítást, vagy egy begin..end utasításblokkot lehet írni a then és az else után. Az else részt akár el is hagyhatjuk, ha nincs rá szükség. Ez a következő kombinációkat adja:

```
if feltétel then utasítás;          // else nélkül, egy elemi utasítás

if feltétel then                    // else nélkül, utasításblokk
begin
      utasítás;
      ...
      utasítás;
end;



if feltétel then                    // else résszel, egy elemi utasítás
      utasítás
else
      utasítás;

if feltétel then                    // else résszel, utasításblokkal
begin
      utasítás;
      ...
      utasítás;
end
else
begin
      utasítás;
      ...
      utasítás;
end;
```

Fontos figyelni rá, hogy nem attól tartozik egy utasítás egy utasításblokkba, hogy mennyi bekezdéssel írod, hanem attól, hogy hova írod:

```
if getPort(1)=1 then
     setPort(12,1);      //--itt vége van az if..then-nek, pontosvessző is van
     setPort(11,1);      //--ez egy különálló utasítás, amit mindíg végrehajt

if getPort(1)=1 then
begin
     setPort(12,1);      //--mindkét utasítás ugyanabban a blokkban van,
     setPort(11,1);      //--a then után, ha a feltétel igaz, végrehajtódnak
end;                     //--itt vége van az if..then-nek, pontosvessző is van
```

Összetett feltételek

Csináljunk láncfűrészt – azaz akkor kapcsoljuk be a ledet, ha a 3-as és a 4-es portra kötött kapcsolók mind be vannak kapcsolva.

```
begin
      while 1=1 do
      begin
            if (getPort(3)=1) and (getPort(4)=1) then
                  setPort(12,1)
            else
                  setPort(12,0);
      end;
end.
```

Nos? Tulajdonképpen csak a feltétel vátozott. Annyiban változott, hogy két getPort van benne, és az ezekhez tartozó összehasonlító kifejezés `(getPort(4)=1)` értékei közé egy ***logikai és*** műveletet teszünk. Ennek a módja: `(feltétel) and (feltétel)`

Természetesen a dolgokat lehet tovább kombinálni: `(feltétel) and (feltétel) and (feltétel) .... stb.`

A következő logikai műveleteket lehet használni:

`(feltétel) and (feltétel)` Akkor igaz, ha mindkét feltétel egyszerre igaz.

`(feltétel) or (feltétel)` Akkor igaz, ha legalább egyik feltétel igaz.

`(feltétel) xor (feltétel)` Akkor igaz, ha pontosan egy feltétel igaz.

`not (feltétel)` Akkor igaz, ha a feltétel hamis.

Például:

`if not (getPort(2)=1) then...` // ha a 2-es gomb nincs lenyomva...

`if (getPort(1)=1) or (getPort(2)=1) then...` // ha az 1-es vagy 2-es gomb le van nyomva...

`if (getPort(1)=1) and (getPort(2)=1) then...` // ha az 1-es és a 2-es gomb le van nyomva...

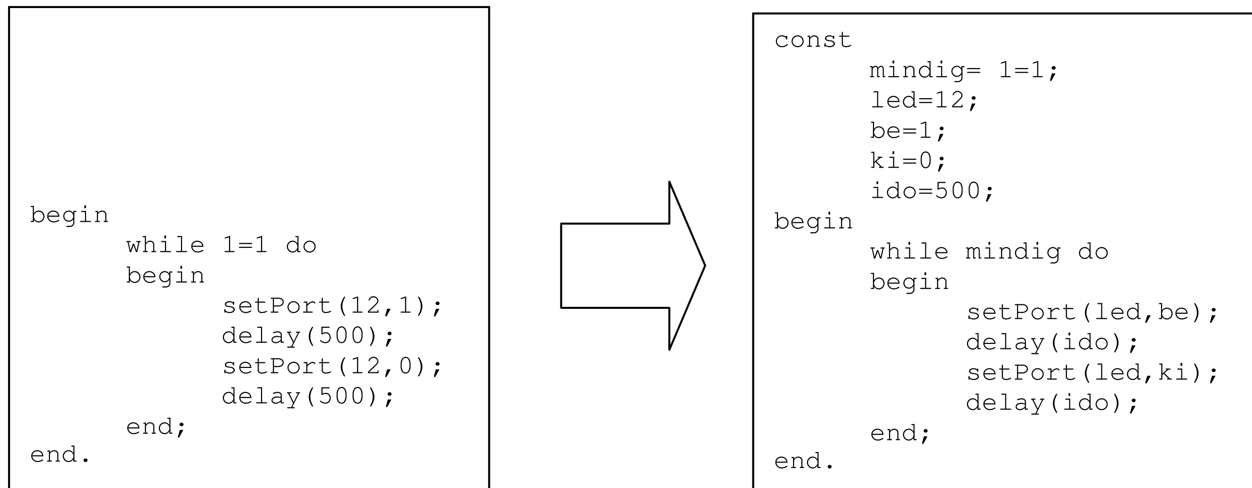
A feltételek kiértékelésének sorrendjét zárójelekkel lehet megadni – mint a matek képleteknél. A feltételek tetszőlegesen kombinálhatók, pld:

`if ((getPort(1)=1) or (getPort(2)=1)) and (getPort(3)=1) then...` // ha (az 1-es vagy 2-es gomb le van nyomva) és a 3-as is be van kapcsolva...

Konstansok

A konstans jelentése állandó. Azért jók a konstansok, hogy könnyen megjegyezhető neveket használhassunk csúnya számok helyett. (Milyen jó is, hogy matek órán csak annyit kell írni: π (pi), nem pedig azt, hogy 3.14159265358979323846264338327950288. Nem jobb? Ha nem jobb, akkor is legalább kevésbé kellemetlen ☺)

A konstansokkal könnyebben olvasható – ezáltal könnyebben megérthető lesz a programunk:



Nos, rövidebb az nem lett. (A Szép program fejezetben már láttuk, hogy attól még ha egy program rövid, nem biztos, hogy egyszerű, vagy áttekinthető. A rövidség nem a program minőségének mértéke.)

Viszont könnyebben áttekinthető lett, jól látszódik, hogy a **`setPort(led,be)`** az bekapcsol egy ledet. Még a **`while mindig do`** is kedvesen mutatja, hogy ez a ciklus bizony mindíg megy körbe-körbe.

Az is praktikus, hogy minden mágikus-szám (olyan számok, amiknek a jelentését most tudod, de két hónap múlva már nem biztos) egyetlen helyre, a program elejére van kigyűjtve. Ha például gyorsabb villogást szeretnél, elég az ido=500-at átírni ido=200-ra, egyetlenegy helyen (nem pedig a két delay-nél, két helyen). Nem csak hogy olvashatóbb lett a program, de könnyebben átalakítható is: hiszen a legtöbb változtatáshoz (melyik porton van a led, milyen gyorsan villogjon) a programhoz nem is kell hozzányúlni, elég a konstansok értékeit átírni.

Deklaráció

A deklaráció azt jelenti: kinyilvánítani, kihirdetni. A programozás során létrehozhatunk tetszőleges névvel olyan dolgokat, amikről korábban fogalma sem volt a számítógépnek. A konstansok esetében például, a számítógépnek fogalma sem volt, mi az, hogy **`be`** – egészen addig, amíg a program legelején a const kulcsszó után meg nem adtuk ennek az értékét.

A program elejét – az első begin előtti részt – nevezzük deklarációs résznek, mert itt lehet mindenféle új dolgokat megismertetni a számítógéppel. Ezek a dolgok lehetnek: konstansok (előző fejezet), változók (következő fejezet), eljárások, függvények (így lehet saját

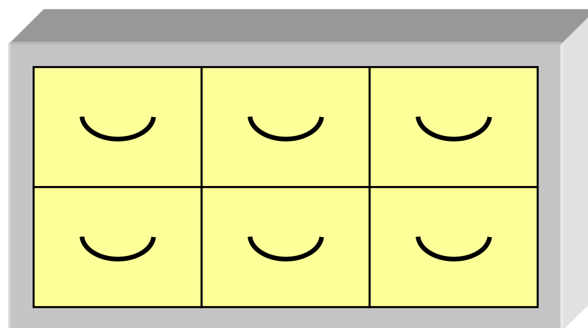
utasításokat csinálni), típusok, és hasonló apróságok. Egyetlenegy dologra kell figyelni: ezen dolgok elnevezésekor nem használhatunk olyan neveket, amik Pascal utasítások (pld. if, then, while), és nem kezdhetjük ezen neveket számokkal (a „12kalapacs" nem jó név egy konstansnak). Nem használhatunk ékezetes betűket, írásjeleket, és szóközt (a „három testőr!" szintén nem jó név). Viszont használhatunk aláhúzást: harom_testor vagy a Szép program fejezetben megismert camelcase módszert: haromTestor.

Olyan neveket jó kitalálni, amik fedik a jelentésüket. Például, ha egy port be-kikapcsolásához csinálsz konstansokat, akkor a `be=1` konstans egyszerű, jól fedi a jelentését, de a `portraKitesszukAVillanytEmiattAktivLesz=1` már kicsit túl sok ☺. A `b=1` viszont kicsit túl kevés: mi az, hogy b? Béget, Belső, Baromfi, Bárány, Birka, Burger?

Változók

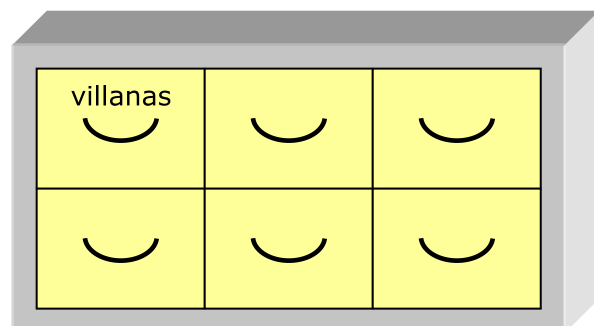
A konstansok nagyon praktikusak – hiszen a mágikus számokat egyszerű, kifejező, jól megérthető szimbolikus nevekkel tudjuk helyettesíteni. Azonban, a konstansok értéke fix, a program futása során nem változtatható. De mi van, ha szeretnénk változtatni az értéküket? Erre valók a változók.

Úgy lehet elképzelni ezt, mintha a számítógépnek lenne egy szekretere (azaz fiókos szekrénykéje). Minden fiókba adatokat lehet betenni, és minden fióknak nevet lehet adni.



Amikor deklarálunk egy változót, akkor kapunk egy fiókot a változó nevével. A változó deklarálása hasonlít a konstansok deklarációjához, azonban itt azt is meg kell adni, hogy mit szeretnénk tárolni a fiókunkban (attól függően, hogy mit akarunk beletenni, más-más nagyságú fiókot kapunk majd).

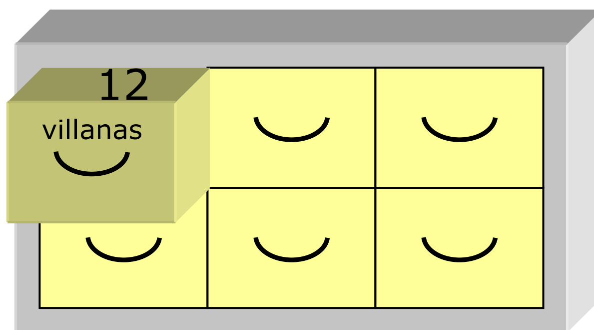
```
var
      villanas:integer;
begin
      . . .
```



A változó deklarációjához tehát csak annyi kell, hogy a var kulcsszó után írjuk a választott nevét, utána egy kettőspontot, majd a változó típusát (azaz, hogy mit szeretnénk tárolni). Először a következő típusokat fogjuk használni: integer – egész szám, single – valós szám, string – szöveg.

A változónak értéket adhatunk az értékadás (kettőspont-egyenlő) művelettel (operátorral):

```
var
      villanas:integer;
begin
      villanas:=12;
      . . .
```



Ekkor beleteszi a gép a villanás nevű fiókba a 12-es számot. Természetesen egy értékadáshoz nem csak fix számok, hanem bármely kifejezés felhasználható, ami olyan értéket eredményez, ami a változónk típusa. Most a változónk egész számokat tárol, tehát szóba jöhet mindaz, amivel számokkal lehet műveletet végezni: összeadás, kivonás, stb... Pld: **`villanas:=12*42-34;`**

Már tudjuk, hogy bárhova, ahova korábban számokat írtunk, írhatjuk egy konstans nevét is, így pld: **`villanas:= ido * 2;`** is teljesen helyén való (előtte persze az ido nevű konstanst deklarálni kell).

Nem csak konstans nevét, hanem változó nevét is írhatjuk egy kifejezésbe – pont úgy, mint egy konstanst. Ekkor a gép megnézi, hogy a változó nevének megfelelő fiókban éppen milyen érték van, és azt veszi elő: pld: **`delay(villanas * 10);`** és máris előszedi a villanás fiókban lévő értéket, így ez delay(12 * 10)-et fog jelenteni.

Ha bárhova, ahol szám van, lehetne írni konstansot, vagy változót, akkor... akkor lehet olyat is írni, hogy **`villanas:=villanas;`** ami bár teljesen helyes, sok értelme nincs, hiszen ez után is pont az lesz a változónk értéke, mint ezelőtt. De, ha egy picit matematikuskodunk, akkor érdekes dolgokat lehet művelni: **`villanas:=villanas+1;`** Ez ugye előveszi a villanás aktuális értékét (ami 12), hozzáad 1-et, így kijön, hogy 13, és ezt visszarakja a villanás feliratú fiókba. Ha ugyanezt az utasítást újra végrehajtjuk, akkor 14, majd 15, stb... lesz, amit kapunk.

Szöveges változók

A szöveges változók is ugyanúgy működnek, mint a számot tartalmazó változók. A deklarálásukkor a string adattípust kell használni. Értékadáskor a szöveges értékeket egyszeres idézőjelbe (aposztróf) kell tenni. A szövegeket nem lehet szorozni, kivonni, osztani, viszont az összeadás művelet működik. Az összeadás művelettel két szöveg-darabka egymás mellé adódik össze, pld. 'alma' + 'fa' eredménye 'almafa'.

A printText utasítással egy szöveget ki lehet iratni a DonGo szerkesztő ablakának aljára:

```
var
      duma:string;
begin
      duma:='alma';
      duma:=duma+'fa';
      printText('Ez lett:');
      printText(duma);
end.
```

Számoljunk a változókkal!

Mire is jók ezek a változók? Oké, leginkább arra, hogy adatokat tároljanak, de mire is lesz ez jó? A legegyszerűbb példa a számolás. Csináljunk egy programot, ami 5-ször villant meg egy ledet. A szokásos ledvillogtató progihoz képest a változásokat kisárgítottam:

```
var
	villanas:integer;
begin
	villanas:=1;

	while villanas<=5 do
	begin
		setPort(12,1);
		delay(500);
		setPort(12,0);
		delay(500);

		villanas:=villanas+1;
	end;
end.
```

Először is, ahhoz, hogy 5-ig tudjunk számolni, kell egy változó. Mivel ez a villanások számát tartalmazza, ezért legyen a neve fapapucs... vagy nem is, inkább villanas.

Először adunk neki egy kezdőértéket, mégpedig 1-et. A ciklus feltételét a szokásos 1=1 helyett (ami mindíg igaz) kicseréltem villanas<=5-re. Ez akkor igaz, ha a villanás változónk értéke kisebb, vagy egyenlő mint 5. (Most még csak 1 az értéke, most adtuk neki, tehát a feltétel igaz.)

A ciklus magjában villantunk egyet a leden, majd növeljük a villanás változó értékét. Így már a villanás változó értéke 2 lesz (eddig 1 volt, 1+1 az... azt hiszem, kettő lesz).

Ezután megint a while feltételénél találja magát a gép, 2<=5, igen, igaz, akkor mégegy kör a villantással, és a változó növelésével. Egészen addig megy körbe, amíg az ötödik villanás után a villanás változót megnövelve 6 lesz a változó értéke. A 6<=5 már nem igaz, így a while nem csinál semmit, megy tovább a program a while utáni utasításblokk utánra – ahol most semmi sincs, csak a programvégi end.

A for-ciklus

Sokszor van szükség arra, hogy egy kis programrészt megadott számúszor ismételjünk meg. Bár erre a while tökéletes, de van egy egyszerűbb módszer: a for ciklus. Ehhez kell egy változó, és a for ciklus ennek a változónak az értékét növeli, amíg el nem éri a kívánt végértéket. Az előző fejezet 5-ször villanó progija for ciklussal:

```
var
      villanas:integer;
begin
      for villanas:=1 to 5 do
      begin
            setPort(12,1);
            delay(500);
            setPort(12,0);
            delay(500);
      end;
end.
```

Szakácskönyv

Ebben a részben gyakran előforduló feladványok megoldásait gyűjtöttem össze. Ha „hogyan-is-kell ezt-és-ezt csinálni” kérdésed van, akkor itt könnyen megtalálod a választ.

Program (hogyan kell kezdeni?)

```
begin
      //--ide jönnek az utasítások
      //--egy sorba egy utasítás, a végén pontossvessző
end.
```

Utasítások folyamatos ismétlése (végtelen ciklus)

```
begin
      while 1=1 do
      begin
            //--az ide írt utasításokat folyamatosan ismétli
      end;
end.
```

Utasítások ismétlése néhányszor (for ciklus)

```
var   szam:integer;
begin
      for szam:=1 to 5 do
      begin
            //--az ide írt utasításokat 5x ismétli
      end;
end.
```

Utasítások ismétlése amíg egy feltétel igaz

```
begin
      while getPort(1)=1 do
      begin
            //--az ide írt utasításokat addig ismétli,
            //--amíg az 1-es port aktív (gomb megnyomva)
      end;
end.
```

Elágazás: egyik vagy másik utasításblokk végrehajtása ha a feltétel igaz

```
begin
      if getPort(1)=1 then
      begin
            //--ezeket az utasításokat hajtja végre,
            //--ha a bemenet be van kapcsolva
      end
      else
      begin
            //--ezeket az utasításokat hajtja végre egyébként
            //--(ha a bemenet ki van kapcsolva)
      end;

end.
```

Kimenet be- vagy kikapcsolása

```
      setPort(12,1); //--bekapcsolja a 12-es kimenetet
      setPort(12,0); //--kikapcsolja a 12-es kimenetet
```

Ha egy bemenet be- vagy kikapcsolt...

```
      if getPort(1)=1 then
      begin
            //--ezeket az utasításokat hajtja végre,
            //--ha a bemenet be van kapcsolva
      end
      else
      begin
            //--ezeket az utasításokat hajtja végre egyébként
            //--(ha a bemenet ki van kapcsolva)
      end;
```

Várakozás megadott ideig

```
      delay(2000); //--vár 2 másodpercet (2000 ezredmásodperc)
```

Várakozás gomb megnyomására (vagy bemenet aktívvá válására)

```
      while getPort(1)=0 do ; //-- a „semmi” utasítást ismétli
```

Várakozás gomb elengedésére (vagy bemenet inaktívvá válására)

```
      while getPort(1)=1 do ;
```

Pascal referencia

Ez a fejezet a Pascal nyelv utasításait és egyéb dolgait foglalja össze. Bármely Pascal-szerű nyelvben (Delphi, Turbo Pascal, Borland Pascal, FreePascal, Lazarus, mikroPascal) ezek mind egyformák.

utasítás; - végrehajtja a megadott utasítást.

Az utasításoknak lehetnek bemeneti paraméterei, ami befolyásolja, hogy mit csinál az utasítás. Pld.: delay(1000); – vár 1000 ms ideig.

Ha több bemeneti paraméter van, azokat vesszővel kell elválasztani: setPort(12,1);

Vannak olyan utasítások, amik eredményt is szolgáltatnak – ezeket függvénynek is nevezik. Az utasítások eredménye felhasználható bárhol, ahova értékeket írnánk: összehasonlításhoz, pld: if getPort(12)=1 then... vagy érték helyett: setPort(12, getPort(1)).

begin

utasítás;

utasítás;

end; - utasításblokk, végrehajtja az utasításokat sorban egymás után.

while feltétel do utasítás/utasításblokk; - amíg a feltétel igaz, megismétli az utasítást/utasításblokkot

if feltétel then utasítás/utasításblokk; - ha a feltétel igaz, végrehajtja az utasítást/utasításblokkot

if feltétel then utasítás/utasításblokk else utasítás/utasításblokk; - ha a feltétel igaz, végrehajtja a then utáni utasítást/utasításblokkot, máskülönben, ha a feltétel nem igaz, az else utáni utasításblokkot hajtja végre.

for változó:=kezdőérték to végérték do utasítás/utasításblokk; - a változó értékét a kezdőértéktől a végértékig növeli, és minden növeléskor végrehajtja az utasítást/utasításblokkot.

const konstansnév=érték; - a megadott értéket hozzárendeli a konstansnévhez. Ezután bárhova, ahova a konstans nevét írjuk, a gép a megadott értéket használja. A konstansok nevének betűvel kell kezdődnie, és ékezetes betűt, vagy szóközt nem tartalmazhat. Olyan nevek, amik már léteznek (a Pascal használja őket) nem használhatóak (pld. if, while, for).

var változónév:típus; - létrehoz egy változót a megadott névvel. A változó adatot tárol, és bármikor tehetünk bele adatot az értékadás művelettel. Bárhol ahol a változó nevét írod, magától a változó értékét használja a gép. Praktikus változótípusok: integer – egész szám, byte: pozitív egész szám 0..255 között, real: valós szám, string – szöveg. A változók nevének betűvel kell kezdődnie, és ékezetes betűt, vagy szóközt nem tartalmazhat. Olyan nevek, amik már léteznek (a Pascal használja őket) nem használhatóak (pld. if, while, for).

deklarálás – változók, konstansok nevének megadása. A program eleje és az első begin közötti részben lehet a const és a var kulcsszavakkal konstansokat és változókat megadni.

:= (értékadás) – ezzel lehet egy változónak értéket adni, pld: szuletesiev:=1984;

matematikai kifejezés – olyan kifejezés, aminek az értéke egy szám lesz. Pld: 4+8*2. Matematikai kifejezés írható bárhova, ahova szám kerülne.

matematikai műveletek – olyan műveletek, amiket számokon lehet végezni, és számokat adnak eredményül.

+	Összeadás, pld. 2+3 az 5.
-	Kivonás
*	Szorzás
/	Osztás (ez valós számot eredményez)
div	Egész osztás (az eredménye egész szám)
mod	Osztási maradék, pld. 14 mod 10 az 4.

összehasonlító kifejezés – olyan kifejezés, ami logikai eredményt (igaz vagy hamis) szolgáltat.

=	Egyenlőség, pld. 2=4 az nem igaz
<>	Nem egyenlő, pld. 2<>4 az igaz
<	Kisebb
>	Nagyobb
<=	Kisebb vagy egyenlő
>=	Nagyobb vagy egyenlő

logikai műveletek – olyan műveletek, amiket logikai értékeken (igaz (true) vagy hamis (false) lehet végezni, és logikai eredményt szolgáltatnak.

not	Invertálás, not (2=2) az false
and	Logikai és művelet, az eredménye akkor igaz, ha mindkét paramétere igaz, pld. true and true az igaz.
or	Logikai vagy művelet, az eredménye akkor igaz, ha bármely paramétere igaz.
xor	Logikai kizáró-vagy művelet, az eredménye akkor igaz, ha csak egyetlen paramétere igaz. Pld. false xor true az igaz

string műveletek – olyan műveletek, amiket szöveg típusú adatokon lehet elvégezni.

+	Szövegek összefűzése, pld. 'alma' + 'fa' az 'almafa'
length('szöveg')	Megadja a szöveg hosszát (betűinek számát)
copy('szöveg',kezdet,hossz)	Kimásol egy darabot a szövegből, a kezdet sorszámú betűtől hossz darab betűt.
[pozíció]	Visszaadja a megadott helyen lévő betűt, pld. valami:='korte'; valami[3] az az r betű lesz

DonGo referencia

Ez a fejezet a DonGo szimulátorhoz elkészített utasítások gyűjteménye (másnéven programkönyvtár, modul, unit) által biztosított utasítások leírását tartalmazza.

Port kezelés (be-kimenetek)

setPort(portszám, érték) – beírja a megadott értéket a megadott portra.

A digitális portoknál 0 jelenti a kikapcsolt, 1 a bekapcsolt állapotot. Az analóg portoknál 0..255 közötti értéket lehet beírni, 0 a legkisebb, 255 a legnagyobb állapot. Ebben a verzióban 12 portot használhatsz, 1..12 közötti sorszámmal.

getPort(portszám) – visszaadja a megadott port értéket.

A digitális portoknál 0 jelenti a kikapcsolt, 1 a bekapcsolt állapotot. Az analóg portoknál 0..255 közötti értéket kapunk vissza, 0 a legkisebb, 255 a legnagyobb állapot.

Ebben a verzióban 12 portot használhatsz, 1..12 közötti sorszámmal.

Hang kezelés

setAudio(hangminta) – elindítja a megadott sorszámú hangmintát.

A 0-ás sorszámú hangminta elindítása kikapcsolja a hangot, 1..9 között pedig a sounds mappában lévő 1.wav, 2.wav, stb. file-okat játssza le. A configAudio utasítással pontosan megadható, hogy melyik sorszámra mit játsszon le.

getAudio – visszaadja az éppen lejátszott hangminta sorszámát.

Ha éppen lejátszik valamit, annak a sorszámát adja vissza. Ha a lejátszás véget ért, vagy semmit sem játszik le, akkor 0-át ad vissza.

configAudio(hangminta,'filenév') – beállítja, hogy egy hangmintához melyik file tartozik.

Tetszőlegesen beállítható, hogy melyik hangminta sorszámhoz melyik hang-file-t játssza le a rendszer. Pld. configAudio(4, 'djreturnsmono.wav') után a setAudio(4) a sounds mappában lévő djreturnsmono.wav file-t játssza le.

A hanglejátszó wav, mp3, ogg, wma formátumu hangfile-okat tud lejátszani. A megadott file lehet a sounds mappában – ekkor csak a filenevet kell megadni:

```
configAudio(4, 'djreturnsmono.wav');
```

A file lehet bármely helyen, ekkor a teljes elérési utat kell megadni:

```
configAudio(4, 'c:\cuccok\hangok\szuperhang.mp3');
```

A file lehet megosztott hálózati meghajtón, megosztás-névvel vagy IP címmel megadva:

```
configAudio(4,'\\csudaszerver\cuccok\1.wav');

configAudio(4,'\\192.168.0.90\media\1.wav');
```

A file lehet Interneten, teljes URL-el megadva hálózati meghajtón, megosztás-névvel vagy IP címmel megadva:

```
configAudio(4,'http://www.digitalismagia.hu/hangok/1.wav');
```

A file lehet nem csak egyszerű file, hanem file stream is (így működnek az Internetes rádiók). A lejátszó a ShoutCast és IceCast rendszereket ismeri, ha a megadott URL egy file stream-re vagy playlist-re utal, akkor azt leszedi és lejátssza:

```
configAudio(4,'http://yp.shoutcast.com/sbin/tunein-station.pls?id=572998');
```

A configAudio segítségével bármennyi hangmintát használhatsz, ha előtte a configAudio-val beállítod a hangmintához tartozó file nevet:

```
configAudio(152,'szuper.wav');

setAudio(152);
```

Beszéd kezelés

setSpeech(hangminta) – elindítja a megadott sorszámú hangmintát.

A configSpeech utasítással pontosan megadható, hogy melyik sorszámra mit mondjon.

getSpeech – visszaadja az éppen lejátszott hangminta sorszámát.

configSpeech(hangminta,'szöveg') – beállítja, hogy egy hangmintához mit mondjon. Például: configSpeech(1,'welcome'); configSpeech(2,'good bye'); és ezek után a setSpeech(1) a welcome-t mondja.

Timer (időzítő) kezelés

setTimer(timerszám) – elindítja a megadott sorszámú időzítőt. Az időzítő kimenete aktív (azaz 1), amíg a beállított idő le nem tellik.

getTimer(timerszám) – visszaadja a megadott időzítő állapotát. A visszaadott érték 0, ha az időzítő nem aktív, és 1, amíg az időzítő aktív.

configTimer(timerszám,idő) – beállítja a megadott sorszámú időzítő idejét. Az időzítő a setTimer-el bekapcsolás után ennyi ideig aktív, majd magától kikapcsol. Az idő milliszekundumban (ezredmásodperc) értendő, 5 másodperc pld: 5000.

Ebben a verzióban négy timer használható, melyek sorszáma 1..4.

Kiírás

print(érték) – kiírja a megadott értéket a DonGo szimulátor ablakának aljába.

printText('szöveg') – kiírja a megadott szöveget a DonGo szimulátor ablakának aljába.