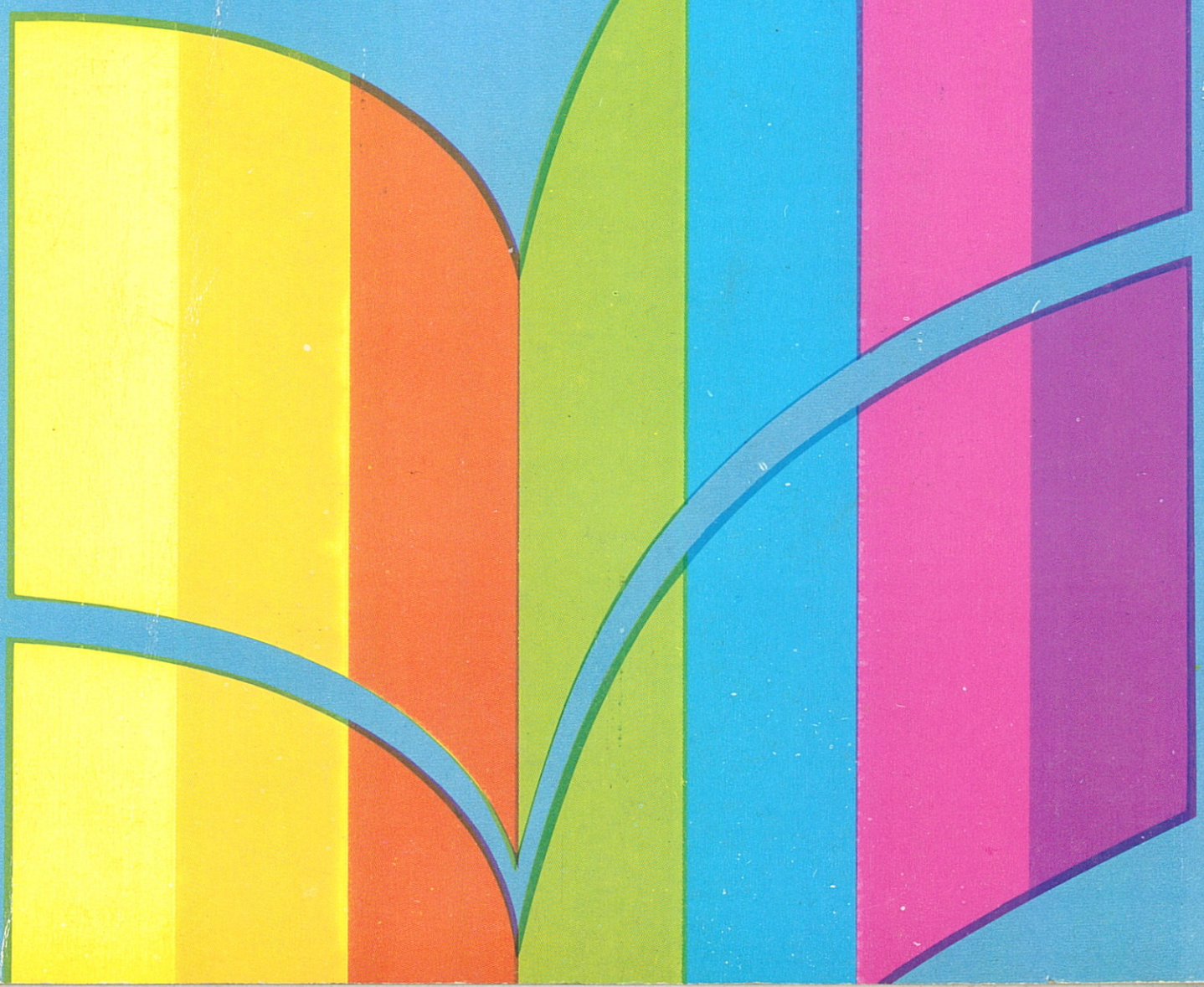
DR. DEÁK LÁSZLÓ

MIKROSZÁMÍTÓGÉPES OKTATÓ-PROGRAMOK KÉSZÍTÉSE ÉS ALKALMAZÁSA

C16
PLUS4
C64
PRIMO
TV COMPUTER

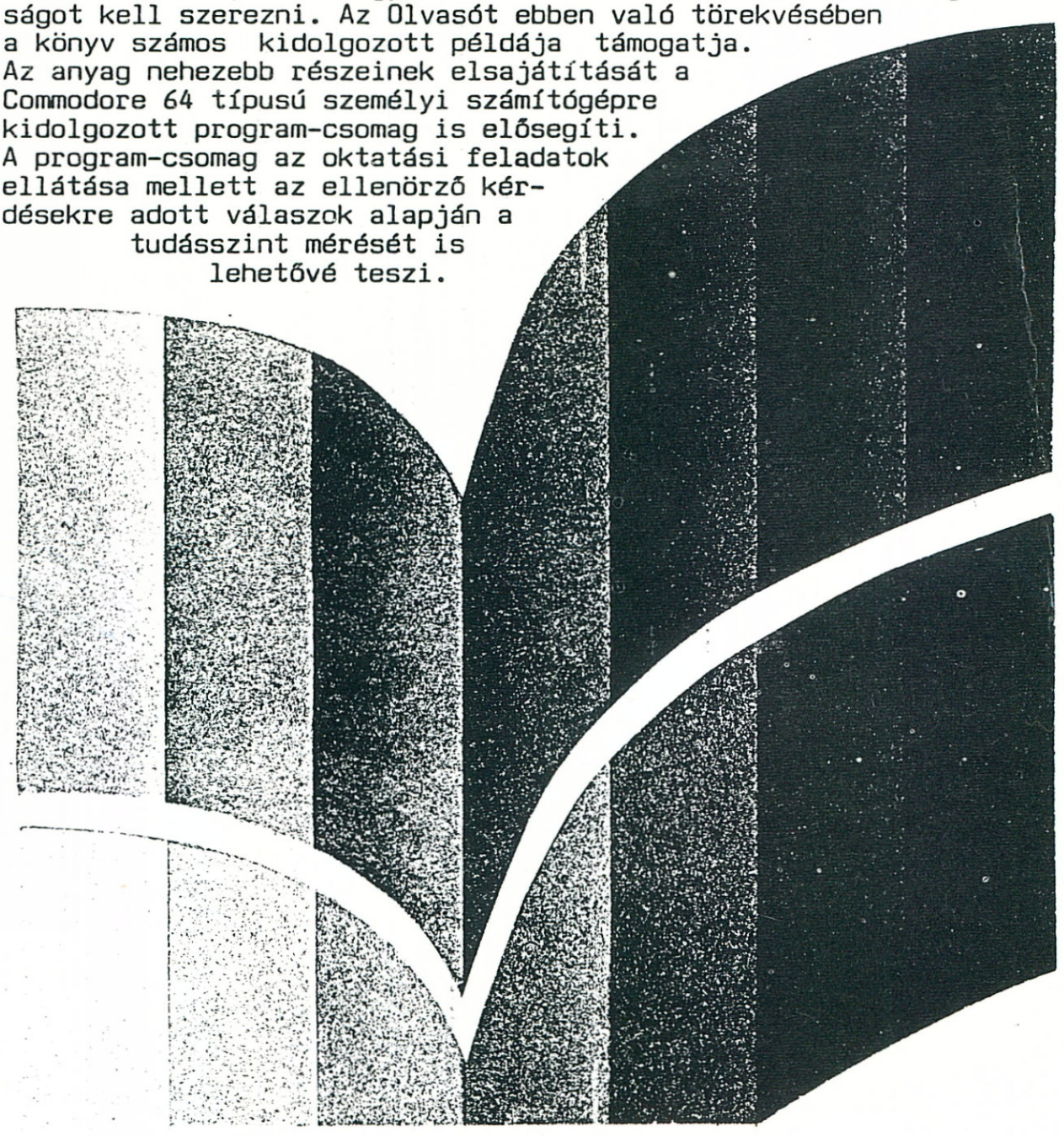
LSI ALKALMAZÁSTECHNIKAI
TANÁCSADÓ SZOLGÁLAT



DR. OBÁDOVICS J. GYULA

MATEMATIKA

A könyv két kötete a középiskolai, technikumi matematika anyagot rendszerezett formában tárgyalja. Segítséget kíván nyújtani a korábban tanult ismeretek felfrissítéséhez, újratanulásához, illetve kibővítéséhez. A " Nyitott Egyetem " matematikai és szaktárgyi anyagának elsajátításához a könyv elméleti anyagának biztos tudása mellett, annak gyakorlati alkalmazásában is kellő jártasságot kell szerezni. Az Olvasót ebben való törekvésében a könyv számos kidolgozott példája támogatja.
Az anyag nehezebb részeinek elsajátítását a Commodore 64 típusú személyi számítógépre kidolgozott program-csomag is elősegíti.
A program-csomag az oktatási feladatok ellátása mellett az ellenörző kérdésekre adott válaszok alapján a tudásszint mérését is lehetővé teszi.



DR. DEÁK LÁSZLÓ

MIKROSZÁMÍTÓGÉPES OKTATÓPROGRAMOK KÉSZÍTÉSE ÉS ALKALMAZÁSA

C16
PLUS4
C64
PRIMO
TV COMPUTER

ALKALMAZÁSTECHNIKAI TANÁCSADÓ SZOLGÁLAT
BUDAPEST, 1988

Lektorálta:

Dr. Lukács József

Dr. Szilassi Lajos
főiskolai docens

Kiadó: LSI Alkalmazástechnikai Tanácsadó Szolgálat
Felelős kiadó: Dr. Kovács Magda
Témafelelős: Sziklai Klára
Szedés: LSI ATSz
Technikai szerkesztő: Nagy Olivér
ISBN: 963 592 751 7
Engedélyszám: 51938
Készült az OMIKK nyomdájában
1011 Budapest, Gyorskocsi u. 5–7.
Felelős vezető: Tóth Károly

TARTALOMJEGYZÉK

BEVEZETÉS

Örvendetes, hogy a magyar iskolák csaknem mindegyike rendelkezik mikroszámítógéppel. A jó programok biztosítása azonban zökkenőkkel halad. Gyakorlott pedagógusoknak ritkán van idejük és lehetőségük arra, hogy programozóvá képezzék magukat. Sok programozó viszont nem ismeri az iskolák igényeit és lehetőségeit, így a kereskedelemben kapható oktatóprogramok nem mindig segítik igazán az iskolai munkát. E könyv szándéka, hogy szerény lehetőségeivel szűkítse a meglévő szakadékot, a két oldalt közelítse egymáshoz.

Matematikus és programozó legfeljebb az első hat fejezetet olvassa el, a többi olyan pedagógusoknak szól, akik csak a BASIC alapelemeit ismerik, de van számítógépük és szeretnének valamit önállóan csinálni. Ezért egy gyakorlott programozó sok mindent furcsának találna, nem értené, miért nem írtuk a programokat szebben, tömörebben, úgy, hogy gyorsabban fusson, kevesebb helyet foglaljon, miért magyarázunk 'egyszerű' dolgokat, stb. Akiknek viszont szól a könyv, ezeket feltehetőleg nem róják fel. Egy pedagógus az iskolai munka mellett, jó ha a BASIC nyelv alapjait el tudja sajátítani, futtatni a kész programokat. Ez a könyv azoknak kíván segíteni, akik az alapokon túl kívánnak lépni, szeretnének testre szabott, saját munkájukat segítő egyedi programokat írni és használni. Megpróbálunk a könyvben olyan modulokat, mintákat adni, amelyekben a szöveget sajátjukra cserélve, s a modulokat összefűzve kész az első használható program. Egy-egy programrészt bemutatunk a 'legkezdetlegesebb' formában, majd fejlesztve, tömörítve, jobb megoldásokat ismertetve. Ebben a folyamatban azonban nem megyünk messzire, alapvető szempont a könnyű megértés. Az egyes megoldásokat többnyire bemutatjuk a legelterjedtebb gépekre (C16, plus/4, C64, PRIMO, TVC), ezért értelemszerűen mindenki talál a könyvben számára feleslegeset - amiért elnézést kérünk.

A könyv első részében az oktatóprogramok történeti tapasztalatai és didaktikája kapott helyet. Úgy gondoljuk, ezzel segíthetünk abban, hogy a készítendő programok fontos, konkrét didaktikai feladatok megoldását tűzhessék ki célul.

A készen kapható oktatóprogramokhoz rendszerint nincs semmilyen felhasználói útmutató, módszertani ajánlás. Bízunk abban, hogy a könyv első része e kész programok alkalmazásában is segítséget nyújthat.

A könyvben szereplő programlisták részben tanárjelöltek oktatására, illetve iskolákban felhasználásra került programokból valók, részben a könyv mondanivalójának illusztrálására készültek. Kazettán is megrendelhetők a könyv végében elhelyezett megrendelőlap beküldésével. A jelölésekben a magyar nyelvű számítástechnikai (részben a pedagógiai) irodalomban megszokottakhoz alkalmazkodtunk.

A könyv elsősorban a nem matematika szakos pedagógusokhoz szól - a szerző sem az. Célunk, hogy reál és humán szakosok egyaránt kedvet kapjanak a programíráshoz, amelyet csak elkezdeni nehéz. Ehhez a kezdéshez kíván sok sikert

Szeged, 1988.

a szerző.

1. Mi az oktatóprogram?

A programozott oktatás kezdeteire vonatkozóan nem egységesek a vélemények. Akik az elgondolás lényegét a gépek alkalmazásában látják, a XIX. századra teszik kezdetét, ugyanis az első oktatógép szabadalmat 1866-ban adták ki egy helyesírást tanító gépre. Akik az egyéni ütemben való haladást tekintik legfontosabbnak, az ókori Görögországig vezetik vissza a gondolatot: ott alakultak ki az egyéni oktatás első formái. Vannak, akik a programozott oktatást a tanulók egyéni munkája egyik válfajának tekintik, nem pedig más módszernek.

A programozott oktatás tulajdonképpeni története azonban 1954-től számítható, B.F. Skinner nagyhatású tanulmányának [1] megjelenésétől. Bár az alapok jóval messzebbre nyúlnak vissza, nem véletlen, hogy a programozott oktatás nagyarányú elterjedése az 50-es évek második felében és a 60-as években következett be. A fejlett országokban ekkor érte el a tudományos-technikai forradalom a közoktatást, és döntő változást sürgetett. A pedagógia egykori célja - átadni az emberiség által alkotott, gyüjtött minden lényeges értéket, ismeretet - végleg tarthatatlanná vált. A legfontosabb kényszerítő hatások:

- Az oktatás tartalmát adó tananyag mennyiségileg és minőségileg növekedett, változékonyabb lett.
- A képzés legtöbb szintjén tömegigénnyé vált a tanulás.
- A követelmények emelkedése ellenére a képzési időt nem lehetett lényegesen növelni.
- A fejlődés egyre gyorsuló tempója miatt a permanens tanulás igénye került előtérbe, nélkülözhetetlenné váltak az átképző és továbbképző tanfolyamok.

A tudományos-technikai forradalom eredményeként technikai téren megindulhatott az oktatás korszerüsítése: egyre modernebb információközlő berendezések jelentek meg az iskolákban. Az új technika azonban nem jelentette a módszerek, szervezeti formák, vagyis a tanítási technológia olyan fejlődését, amely megfelelt volna a megnövekedett társadalmi igényeknek. Mind jobban kiütköztek a hagyományos oktatási módszerek fogyatékosságai, melyek alapvetően egy szempont köré csoportosíthatók: a tanulót passziv befogadónak tekintik, a tananyag csak célként funkcionál, eszközként nem. A szakirodalomban leggyakrabban felvetett hiányosságok: a tanítási órákon túlsúlyban van a tanár tevékenysége; túl nagy a szerepe az averziv ellenőrzésnek, kényszerítésnek; a tanítási órákon kevés a megfelelő visszacsatolás és megerősítés; a tananyag megértése, megtanulása és ellenőrzése időben túlzottan szétválik; az oktatási rendszer túlzottan épít az otthoni tanulásra.

A fentiek elemzése alapján kirajzolódott a jövő iskolájának alapvető feladata: **tanulni** tanítson meg, készítse fel az ifjúságot a permanens önművelésre, továbbképzésre, művelődésre, alkotásra. Ennek megvalósítása érdekében fel kell számolni az averziv módszerek túlsúlyát, a tanulókat csak befogadó közegnek tekintő szemléletet, csökkenteni kell az otthoni tanulás szerepét, meg kell teremteni az iskolai munka individualizálásának lehetőségét - csak a legfontosabbakat említve. Érthető, hogy amikor a programozott oktatás ezekre a problémákra kínált megoldási lehetőséget, nagy figyelmet keltett.

A programozott oktatás pszichológiai alapjai E. L. Thorndike tanuláselméletéig [2] nyúlnak vissza, aki állatkisérletekre alapozva új módon tudta megközelíteni a tanulás és gondolkodás fogalmát. Tanuláselméletét három alaptörvényben foglalta össze:

- A gyakorlás törvénye (The Law of Exercise) szerint a gyakorlás révén az inger és reakció kapcsolata erősödik, nem gyakorlás révén gyengül.
- A következmény, vagy effektus törvénye (The Law of Effect) szerint az a kapcsolat, melyet a dolgok megnyugtató állapota kisér, ezáltal erősödik, az viszont, amit kielégületlenség kisér, gyengül.
- A készség törvénye (The Law of Readness) szerint ha egy cselekvésre nem vagyunk felkészülve és mégis kényszeritenek rá, akkor az számunkra kellemetlen.

Thorndike tanulási törvényei elég mechanikus szemléletűek,de a megerősités fogalmának megalkotása és az emberi tanulásra való alkalmazhatósága miatt előremutatóak, továbbfejleszthetőek.

A programozott oktatás első gyakorlati úttörőjeként S. L. Presseyt ismerik el, aki 1926-ban olyan mechanikus vizsgáztatógépet szerkesztett, amely csak a helyes válasz megtalálása után adja a következő kérdést.

A társadalmi szükségletek akkor még nem igényelték a változást, igy gépe feledésbe merült. A 60-as években azután sok olyan elektromos feleltető gépet forgalmaztak, melyek semmivel nem nyújtottak többet Pressey gépénél.

Skinner a pavlovi feltételes reflex egyszerű inger-válasz kapcsolatát, a klasszikus kondicionálást fejlesztette tovább operativ (vagy instrumentális) kondicionálássá. Skinner szerint a helyesen beállitott megerősitésekkel kell elérni a magatartás pozitiv irányú befolyásolását. A tanulás tehát az erősitő mechanizmusok tudatos létrehozása, a tanulói magatartás és e magatartás következményei közötti kapcsolatok által jön létre. A skinneri lineáris programok pszichológiai alapelvei:

1. A tanulásnak állandó aktivitásra kell épülnie.
2. A tanulás sikerét szüntelenül ellenőrizni és megerősiteni kell; ez tartja fenn a motiválást.
3. A feladatmegoldásokat apró logikai lépésekkel, a kérdések minél egyszerűbb fogalmazásával kell megközeliteni.

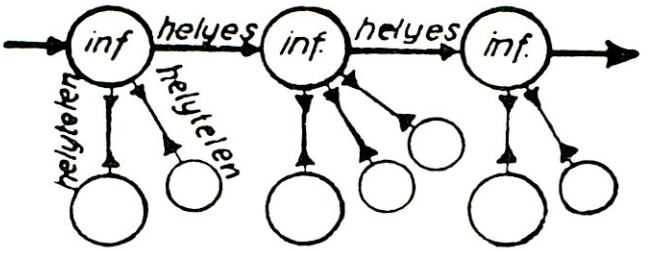
A skinneri tanuláselmélet 1. és 2. alapelve önmagában ma is helyesnek fogadható el, de Skinnernél a megerősités a cselekvés sikerére vonatkozik, a gyors megtanulást azonositja a megértéssel.

A skinneri lineáris programban a tanuló egy-egy információ után kérdést kap, s ha válaszolt megkapja a megerősitést, majd a következő információt. Bármilyen válasz esetén tovább lehet haladni, de a kérdések megfogalmazása olyan, hogy csaknem mindig sikerül jól válaszolni. Az eredeti skinneri programoknál a kérdéseket rendszerint hiányzó szavak pótlása, vagy egyéb kiegészitést igénylő feladat jelenti.

Skinner tanulásalméletét és programozási technikáját sokan birálták. N. A. Crowder kiemelte a helytelen válaszok szerepét: a tanulásban, mint kommunikációs folyamatban a tanulóhoz jutó visszajelentések ellenőrzik, szabályozzák a folyamatot, s ebben nélkülözhetetlen a helytelen válaszok szerepe [3].

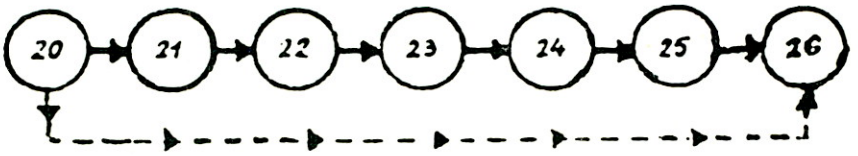
A Crowder által megalkotott elágazásos program kérdéseit feleletválasztásos formában teszi fel. Jó válasz esetén lineáris programhoz hasonlóan haladhat tovább a tanuló.

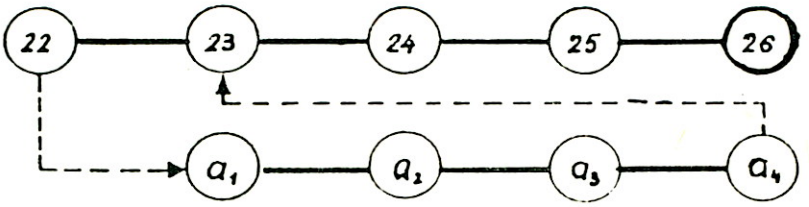
Helytelen válasz után magyarázatot kap hibájáról, majd újra kell válaszolnia. A program sémája:



A hibás válaszok kiigazításának indokolása jobban eligazítja a tanulót, mint a skinneri merev sor. A helyes válasz felismerése azonban nem követel valóban önálló, produktiv értelmi tevékenységet, és tekintetbe kell venni a véletlen eltalálások lehetőségét is.

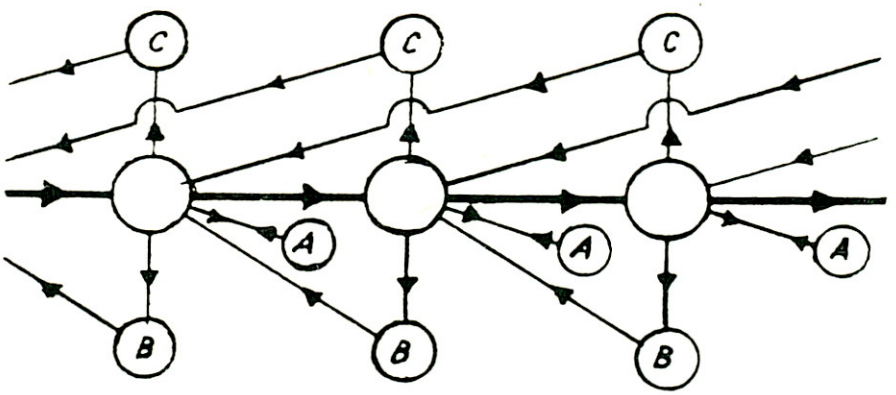
Az oktatóprogramok két fő típusát sokan fejlesztették tovább. A lineáris programokba a jobb tanulók számára beiktattak olyan lépéseket, melyeket átugorhatnak, illetve a program főágába nem tartozó kiegészítéseket:



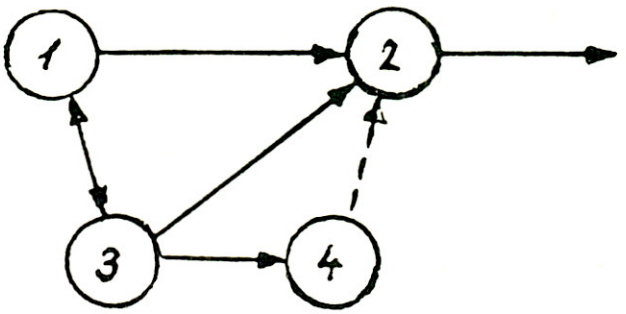


Az elágazásos programtípus lényegesen nagyobb fejlesztési lehetőséget rejt magában: a ma használatos számítógépes oktatóprogramok is innen vezethetők le. A visszautaló (backward branching) program hibás feleletválasztás esetén visszaküldi a tanulót egy

korábbi információhoz:

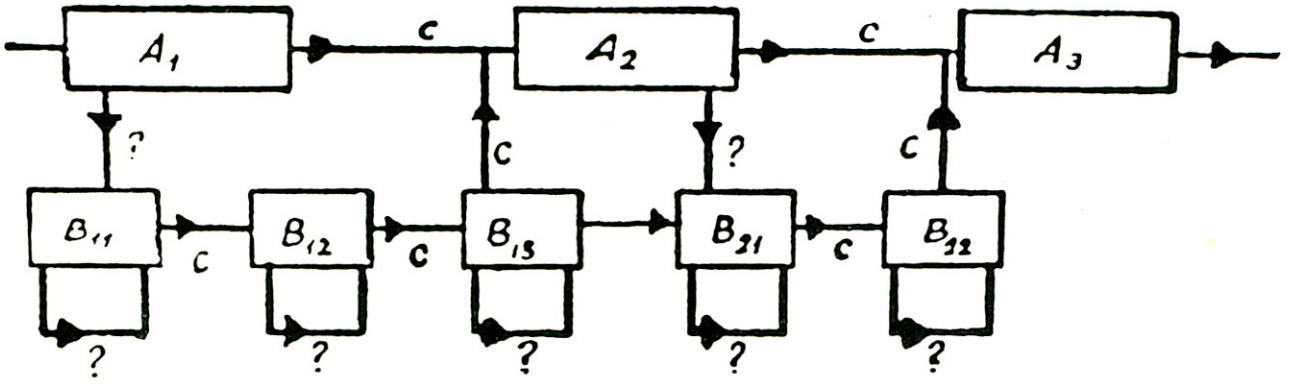


Az előrehaladó módszernél (forward branching) hibás válasz esetén mellékágra jut a tanuló, ahol apró lépésekben, több magyarázattal, de továbbra is előre halad:



Az ugró előrehaladásos (skipbranching), vagy sheffieldi módszer az adott lehetőségek engedte maximális adaptivitást igyekszik megvalósítani. A tanuló önállóan szerkesztett válaszát összehasonlítja a program által megadottal, s ha jól válaszolt, tovább halad a főágon. Rossz válasz esetén mellékágra tér át, ahol kisebb lépésekben kapja a tananyagot. Az itt elhelyezett kritérium-kérdés dönti el a főágra való visszatérés lehetőségét.

A programtipus blokksémája:



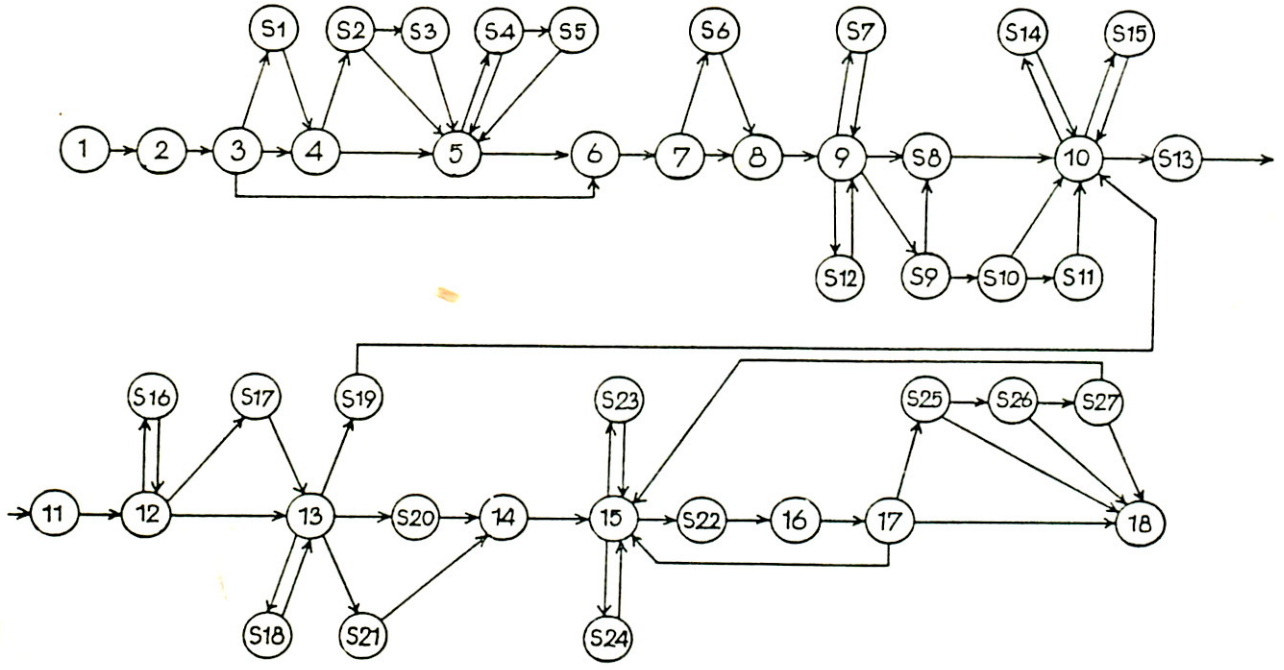
A nyomtatott szövegként elkészithető oktatóprogramok fejlődése itt megállt, a nagyobb adaptivitás eléréséhez már gépekre volt szükség.

A tanulói munka nyomonkövethetősége felé tett lépést egy a szerző által kidolgozott programtipus, amelyet a kémiaoktatás több szintjén kipróbáltak [4,5,6]. Az elágazásos programtipussal szemben az alábbi követelményeket fogalmaztuk meg:

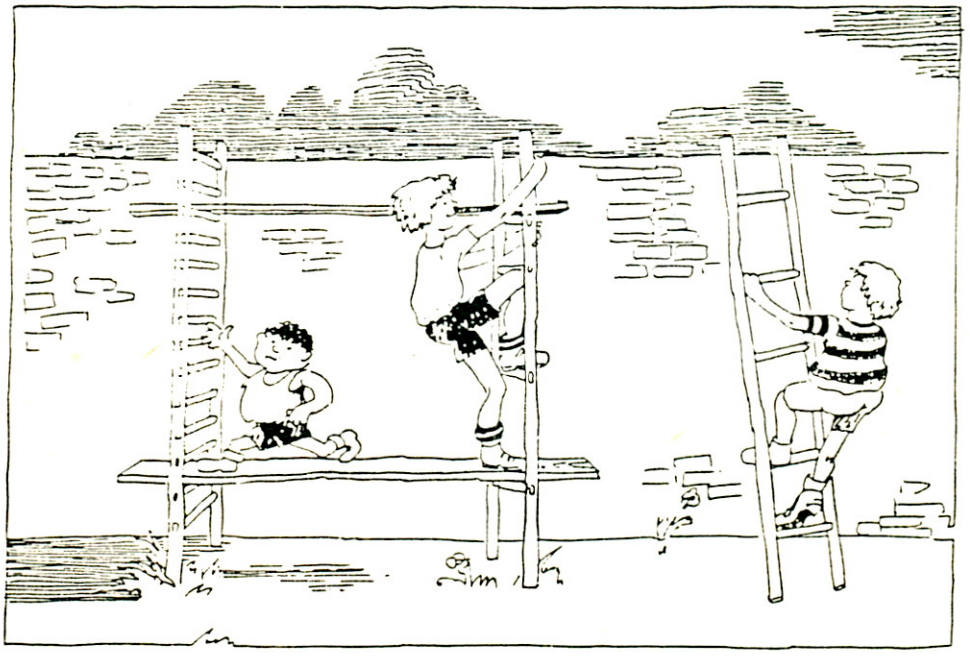
- biztositson önálló feleletalkotási lehetőséget,
- minden lépés után biztositson önellenőrzési lehetőséget,
- regisztrálja a téves válaszokat és a segitség igénybevételét,
- válaszadás előtt a tanuló ne láthassa a válaszokat elbiráló magyarázatokat és a kisegitő információkat.

A vázolt követelményeket biztositó technikai megoldás lényegében a program két részre bontásával oldható meg: az egyik rész tartalmazza egyszerű nyomtatott lapokon a közléseket, kérdéseket - a program fő ágának nagy részét. A másik rész, a "segitő" a helyes és helytelen válaszok magyarázatait és a szükség esetén igénybe vehető kisegitő kérdéseket, magyarázatokat tartalmazza. A segitő teljes szövegét a sorszámok kivételével egy ráragasztott üres papirlap takarja, amely a sorszámok mellett perforálva van. Miután a tanuló eldöntötte, hogy az alternativ lehetőségek közül melyiket választja, illetőleg milyen segitséget kiván igénybe venni, ceruzaheggyel felszakitva a perforációt elolvashatja a szöveget. Önálló feleletalkotás esetén a letakart szöveg a válasz lényeges jegyei alapján küldi tovább a megfelelő lépéshez a tanulót. Az utólag összeszedett lapokból pontosan megállapitható a tanuló programbeli útja, és természetesen értékelhető is.

Egy ilyen program hálódiagrammja:



Az eddig említett programtípusok közül néhány pontosan idézve olvasható az akkori magyar szakirodalomban – tanulmányozásuk segítheti a számítógépes oktatóprogramok készítését is [4–15]. A két alapvető programtípus – lineáris és elágazásos – közötti különbséget jól jellemzi az alábbi rajz, ahol a feladat: felmászni a fal tetejére:



A programozott oktatás gyors terjedése sok elektromechanikus oktatógép megjelenését hozta magával [15,16]. A több magyar oktatógép, illetve visszacsatoló berendezés közül talán a DIACORR BYO–02 egyéni oktatógép érdemel említést. Sajnos a gyakori meghi–

básodások és a javíthatatlanná válás miatt csak rövid ideig használhattuk ezeket a gépeket [6].

A nagyobb egyetemek, kutatóközpontok már a 60-as években rendelkeztek nagyszámítógépekkel – kézenfekvő volt ezek bevonása a programozott oktatás további fejlesztésébe. A számítógép képes volt a tanulót egész munkája alapján folyamatosan értékelve a sokirányba elágaztatott program megfelelő nehézségű részeire vezetni, önállóan alkotott feleleteket is értékelni és hosszú programot, sok adatot tárolni. A két legismertebb ilyen rendszer a CAI és a PLATO [16,17]. Mint a továbbiakban látni fogjuk, tulajdonképpen a mai mikroszámítógépek egy-egy tanulóra vonatkoztatva ugyanolyan eredményesen használhatók, mint az iskolák számára ma is elérhetetlenül drága nagyszámítógépes rendszerek.

A programozott oktatás történetének első évtizedében egyértelműen a sikerekről olvashattunk a szakirodalomban, majd megjelentek a problémák, ellenvetések. A 70-es években úgyszólván 'eltűnt' a programozott oktatás és csak az 'utódairól' lehet már olvasni.

A pszichológiai alapokat egyre többen vitatták, ugyanakkor nem lehetett letagadni a sikereket. Ez a látszólagos ellentmondás arra utalt, hogy a sikerek gyökere talán nem az alapelvekben rejlik. Az érdeklődés a programkészítő tevékenysége felé fordult, és kiderült, hogy eddig soha nem elemezték ilyen alaposan a megtanítandó tananyagot, nem készítettek ilyen pontos elő- és utótesztet, nem próbálták ki, módosították ennyiszer a tanítási szövegeket. Egyre több tapasztalat utalt arra, hogy a programok összeállításánál alkalmazott elemző munka a hagyományos tankönyvek elkészítésében is hasznos [18].

A szorosan vett programozott oktatás története ezzel két évtized alatt lezárult, tapasztalatai és hatása azonban óriási jelentőségűek a pedagógia további története szempontjából. A számítógépprogramokat váró kedves olvasók türelmét kérve feltétlenül indokolt, hogy röviden áttekintsük azokat a területeket, melyek léte, illetve fejlődése jórészt a programozott oktatásnak köszönhető.

Az ókorban és a középkorban megszokott volt az individualizált oktatás – természetesen csak nagyon szűk réteg számára. A modern kor osztálytanításának fellazítására, az individualizálásra számos törekvést ismer a szakirodalom, elsősorban az amerikai kísérleteket: Dalton-Plan, Winnetka-Plan. A programozott oktatás megjelenése ezeknek a törekvéseknek újabb lehetőséget jelentett, de kifulladása nem érintette őket. A két legismertebb, széles körben alkalmazott amerikai project az IPI és a PLAN.

Az IPI (Individually Prescribed Instruction = egyénileg előírt oktatás) lényege a tantervi témák sok egységre bontása, s az egységekhez kidolgozott részletes követelményrendszer. Előteszt alapján jelöli ki a pedagógus az önállóan feldolgozandó tanulási egységeket, majd a záróteszt alapján a következőt, és így tovább.

A PLAN (Program for Learning in Accordance with Needs = igényeknek megfelelő tanulás programja) alapeleme a mintegy kéthetes tanuláshoz szükséges tananyagot, feladatokat, teszteket tartalmazó modul, melynek feldolgozása során váltakozhat az egyéni és kiscsoportos foglalkozás, a tanári konzultáció.

Pszichológiailag megalapozott tananyagfeldolgozási módszer a strukturális kommunikáció. Kidolgozói, az angol Hodgson és Bennett alapnak tekintik Bruner tanuláselméletét [19]. Céljuk a megértés struktúrájának átadása, ennek érdekében az individualizálást a tanulás üteméről kiterjesztik a tanulási stratégia szabad megválasztására. Minden lépésben arra törekszenek, hogy a linearitás korlátait kiküszöbölve a tanulók átlássák az egész téma strukturális komplexitását [17].

Hazai vonatkozásban három fő területen fedezhetjük fel elsősorban a programozott oktatás utóhatásait : az oktatócsomagok, a munkatankönyvek és az iskolai számítógépprogramok megjelenésében.

Az oktatócsomagok tulajdonképpen taneszközegyüttest jelentenek, amelyek egy tananyagrész feldolgozását biztosítják többféle információhordozó (munkalap, film, dia, fólia, stb.) és javasolt feldolgozási menet, módszertani útmutató segítségével. Elég sok oktatócsomag látott napvilágot a központi támogatások segítségével, de tapasztalatunk szerint a pedagógusok inkább egy-egy információhordozó részegységüket használják csak a tanítási órákon.

A munkatankönyvek gyorsan elterjedtek világszerte, hazánkban is több általános iskolai tankönyv készült ebben a formában. A 'hagyományos' programozott oktatás egyes jellemzői leginkább itt fedezhetők fel. A munkatankönyvben egymást váltják a hagyományos leíró részek, a lineárisan és elágazóan programozott részek. Így biztosítható a módszerek változatossága, az önálló tanulói munka és bizonyos mértékig az egyéni haladási ütem. Egy jól szerkesztett munkatankönyv logikai egységei: közlés – kérdés – megerősítés. A tanuló önálló válasza után ügyesen rejtve megtalálja a megerősítést – vagy a tanár közösen ellenőrzi a válaszokat és adja a megerősítést. Egy rövid idézet a 8. osztályos kémia munkatankönyvből [20]:

Hasonlítsuk össze a képleteket!

b_2) Karikázd be a propán képletében azt a részt – atomcsoportot – amellyel nagyobb a molekula az etánmolekulánál!

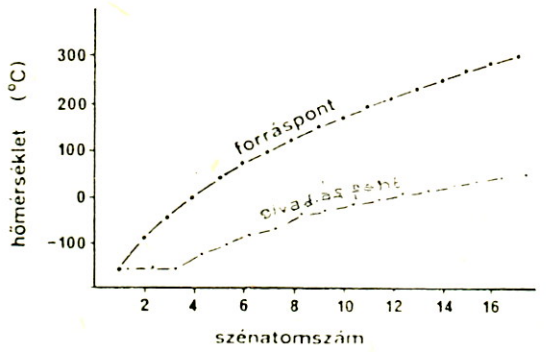
b_3) Karikázd be a bután képletében azt a részt, amellyel nagyobb a molekula a propánmolekulánál!

A telített, nyílt láncú szénhidrogének molekuláit növekvő szénatomszám szerint sorba állítva a molekulák egymástól azonos – CH_2 – **atomcsoporttal** különböznek, **sorozatot** alkotnak. A sorozat tagjainak összefoglaló neve: **paraffinok.** A sorozat első négy tagja: a **metán,** az **etán,** a **propán** és a **bután.**

---

b_4) Számítsd ki, hány g egy mol etán, egy mol propán, illetve egy mol bután!

Tanulmányozd az alábbi grafikont! A nem elágazó láncú paraffinok olvadás- és forráspontját ábrázolja a szénatomszám növekedésének függvényében.



b_5) Milyen szabályszerűséget látsz a grafikonon?

.....

.....

.....

.....

b_6) Állapítsd meg a grafikonról, hogy szobahőmérsékleten melyik szénatomszámú paraffinok a

légneműek: .....

cseppfolyósak: .....

szilárdak: .....

A **paraffinok** tulajdonságai a szénatomszám növekedésével együtt változnak: nő a sűrűségük, az olvadáspontjuk, a forráspontjuk. A sorozat első négy tagja szobahőmérsékleten *gáz*, további tizenkét tagja *cseppfolyós*, a többi *szilárd* halmazállapotú.

Természetesen sok buktatója van a munkatankönyveknek – de minden más önálló munkára alapozott módszernek is. A magyar iskolákban a legélesebb ilyen gond a rossz olvasási készség és a nem elégséges motiváció, nyersebben: a gyerekek nem akarnak tanulni és nem tudnak értelmesen, összefüggően olvasni. Általános tapasztalat szerint egyre kevesebbet tanulnak otthon a gyerekek, tehát a mai magyar iskolának a tanórai elsajátításra kell koncentrálnia – ebben nyujthatnak többet a munkatankönyvek. Az oktatási célú számítógépprogramokban felfedezhetjük a programozott oktatás szinte minden előremutató jegyét : tananyag kis lépésekre bontása, egyéni ütem, önálló feleletalkotás, megerősítések, egyéni haladási ütem, stb. Ezekhez hozzájárultak az eddigiekben vázolt tapasztalatok, és természetesen a sokat tudó, de olcsó mikroszámítógépek elterjedése. A következő fejezetben ezen új lehetőségeket elemezzük.

2. Mit adhat a mikroszámítógép az oktatásnak?

Az előző fejezetben láttunk példát a pedagógiában jellegzetes túlzásra: a programozott oktatás mindent helyettesitő felfogására, majd eltemetésére. Hogy a mikroszámítógépeket ne érje ez a sors, arra két biztató tényezőt találhatunk:

- az oktatáson kivül még igen sok területen terjednek e gépek,
- a pedagógia tudománya és gyakorlata okult a programozott oktatás történetéből.

A bajokat megelőzendő törekedni kell a mikroszámítógépnek, mint sokoldalú oktatási eszköznek a pedagógiai technológia átfogó értelmezésébe való beillesztésére. Feltétlenül káros az a próbálkozás, amely mindenáron bizonyitani akarja a számítógépekkel támogatott oktatás nagyobb hatékonyságát az egyéb tényezők figyelmen kivül hagyásával. Sajnos a magyar oktatásügyben mindig is jellemző volt egy-egy - egyébként fontos - tényező túlhangsúlyozása, ami rendszerint torzulásokhoz vezetett. Ezért már a fejezet cimében is szeretnénk hangsúlyozni, hogy a mikroszámítógép a tanitáshoz **hozzáadhat** valamit, de nem helyettesiti, nem pótolja a többi módszert.

Ebben a fejezetben áttekintjük azokat a tulajdonságokat, lehetőségeket, melyeket kizárólag a mikroszámítógép biztosithat az oktatásban, illetve azokat, melyekben többet nyújthat más eszközöknél, módszereknél.

Az **interaktivitás** a BASIC programnyelv egyik - a tanitás számára - talán legfontosabb tulajdonsága. Annyit jelent, hogy a gép kezelője (=tanuló) a program futása közben beavatkozhat, válaszokkal, adatokkal befolyásolhatja a program futását, vagyis állandó párbeszédben áll a géppel. Valamilyen természeti jelenséget szimuláló programnál pedig a számítógép kizárólagos adottsága a bemenő adatok, feltételek tetszőleges választása, megváltoztatása révén a jelenség más és más lefolyásának bemutatása. Az interaktivitás körébe tartozik a tetszőleges beszédstilus alkalmazása. Egy párbeszédrészlet a fentiek illusztrálására:

"- Szeretettel üdvözöllek Kedves barátom, kérlek, ird le a nevedet !
- Balogh István
- Nagyon örülök Pista, hogy éppen Veled kezdhetem a munkát. A viz párolgásának folyamatát fogod látni. Kérlek, ird be, hány Celsius fokos vizet kivánsz párologtatni a 18 Celsius fokos szobában.
- 80 .
- Nagyszerűen választottál, látni fogod, hogy milyen gyorsan párolog az ilyen meleg viz."

A tanulás kibernetikai modellje az irányitásnak két tipusát különbözteti meg:
- A vezérlés előre meghatározott utasitások alapján történik, nem veszi figyelembe az irányitott rendszer változásait. Ilyen a lineáris oktatóprogram is.
- A szabályozás figyelembe veszi az irányitott rendszer egyes jellemzőinek változásait, és ennek megfelelően irányit(pl.hűtőszekrény, gázkonvektor hőmérsékletszabályozása). A szabályozás fontos eleme a visszacsatolás, amellyel az irányitó rendszer információt kap az irányitott rendszer állapotáról. A jól kivitelezett elágazásos oktatóprogram (elsősorban oktatógéppel) korlátozott mértékben szabályozást tud megvalósitani. A mikroszámítógép lényegesen magasabb fokra emeli a szabályozás lehetőségét, a programkészitőn múlik, hány visszacsatoló paramétert tud figyelembe venni a továbbhaladási irányok eldöntésekor. Egy azonban biztos : a pedagógus számára a gyerekek aktivitása, válaszai, mozdulatai, stb. olyan sokparaméteres visszacsatolást

jelentenek, amelyet semmilyen számítógépprogram nem képes utánozni. Az is igaz viszont, hogy a számítógép nem fárad el, nem lesz ideges, túlterhelt...

Az adaptivitás - alkalmazkodó készség - a számítógépen megvalósított elágazásos oktatóprogramok egyik nagy előnye. Sok paramétert figyelembe vehet a program a soron következő feladatok, kérdések kiválasztásánál, a továbbhaladási irány eldöntésénél. Megfelelően szerkesztett program jól tud differenciálni: a legjobbak néhány lépéssel eljuthatnak a megoldásig, mig a gyengébbek részére akár több száz lépést is biztosit ugyanaz a program. Az adaptivitás olyan látszólag kevéssé fontos kérdésekre is kiterjedhet, mint a tanuló által választható betűméret, képernyőszin, ábrázolási mód, hangeffektus, stb.

A tanulók feleleteinek értékelése a programozott oktatás különböző adathordozókon történő megvalósitásainál egyaránt problematikus volt. A számítógép lehetőséget ad feleltválasztás esetén korlátlan számú alternativára - bár egy ésszerű mértéken túlmenni nem érdemes. Az önálló feleltalkotás lehetőségében minőségi ugrást jelent a számítógép : számadatokat, képleteket, szavakat, mondatokat, képes összehasonlítani a memóriájában tárolt sok variációval, sőt a válasz egyes elemeire vonatkozóan is elvégezheti az összehasonlítást. Természetes lehetőség többféle válaszra külön-külön magyarázatot adni. A számadatok esetében változatos kerekitési, pontossági követelményt adhatunk meg. Az egyes kérdésekre adott válaszok pontértékeit, vagy egyéb jellemzőit tárolhatjuk és bármikor összesitve értékelhetjük a géppel dolgozó tanuló munkáját. Mivel a legegyszerűbb mikroszámitógép is sok adat tárolására képes, igy a munka értékelése nagyon sokrétű lehet (az itt vázolt lehetőségeket a későbbiekben részletes példákon tárgyajuk).

A tanitás szempontjából igen fontos jellemző az egyes mikrogépek grafikai képessége. A grafikus megjelenitési lehetőségek olyan pluszt adnak a mikroszámitógépeknek, amely más tanitási eszközzel nem érhető el. A képernyőn mozgatott ábráknak sajnos a legjobb gépnél is vannak korlátai - ezért egyszerű mozgást sokkal szebben mutathat egy animációs film. A számitógépes grafika előnyét az interaktivitásnál lényegében emlitettük. Elég a kedves olvasó által is biztosan ismert kalandjátékokra (autóverseny, sport-,háborús játékok, stb.) utalni: a képernyőn látható mozgások sok paraméterét a játékos befolyásolja a gép billentyűzetéről, vagy joystickkel (botkormány). A mikrovilág, vagy a nagy sebességű folyamatok számos jelenségét lehet úgy modellezni, hogy a felhasználó dönti el a különböző paraméterek kezdőértékét, változásait. De egy történelmi csata is szimulálható hasonlóan - s a gyerekek között akadhat olyan jó stratéga, aki számitógépe képernyőjén megnyeri a mohácsi csatát!

Jelentős szerep juthat egyes programokban a szineknek.. Az egyes számitógépek ismertetői szeretnek egymásra licitálni a szineket illetően (C 16: 121 szin!), az igazság az, hogy csak igen kevés szint lehet igazán jól használni. A későbbiekben leirt "szinek programja" futtatásáról érdemes statisztikát készíteni: hány szinkombináció nyújtott élvezhető látványt! E kritika különösen igaz az esetek többségében alkalmazott kommersz TV-készülékekre - viszont el kell ismernünk, hogy a drága szines monitorokon sokkal jobb a szinhatás.

Tanitási célú programoknál általában elég, ha kellemes, megnyugtató szinekre törekszünk, esetleg a válaszok elbirálásánál a szinnel nyomatékositunk. Tartalmi jelentőségű kérdéssé léphet elő a szin egyes képletek, ábrák, rajzok részeinek kiemelésénél, a figyelem irányitásánál, vagy rajz-művészettörténet órán használt szindinamikai oktatóprogramban.

A legelterjedtebb mikroszámítógépek mindegyike tud hangot előállítani, némelyikük teljes értékű szintetizátorként is alkalmazható. Mire használjuk a gépek kitűnő zenei adottságait, amikor nem az ének-zene oktatásához készítünk programot? Lehet dicsérő, vagy elmarasztaló hatású kis dallamokat bejátszani; valamilyen ismert zenével jutalmazni; a tartalomhoz kapcsolódó zörejeket, hangeffektusokat alkalmazni, stb. Egy a lényeg: ne terelje el a figyelmet, valamilyen módon a program didaktikai célját szolgálja.

Eddig azt vizsgáltuk, mit adhat a kikroszámítógép a tanításnak. A fejezet befejezésképpen térjünk át arra, hogy **mit nem adhat?**

Egy program lehet tökéletes program mivoltában: preciz, türelmes, igazságos, szép, stb. Egy nem lehet: emberi. A pedagógust segítheti, de nem pótolhatja. A tanítási módszereket váltogatni kell, mert unalmassá válnak, megszűnik a motiváció. Minden módszert gyakran kell felváltson a pedagógus élő szava. A számítógépek jelenlegi erős motiváló hatása csak a mérsékelt és célzott "adagolással" tartható fenn. Nincs arra remény, hogy minden tanulónak külön gépe legyen az osztályteremben, ezért nem is érdemes teljes tanítási egységeket programban feldolgozni (bár néhány iskola egy termében ez elérhető, illetve az lesz).

Hogy mit célszerű programban feldolgozni, milyen célra és hogyan - ezzel foglalkozunk a további fejezetekben.

3. Mire használhatjuk a mikroszámítógépet az iskolában?

A címbeli kérdés megválaszolásához át kell tekintenünk az iskolai tanítás-tanulás részleteit, vagyis az oktatási folyamat szerkezetét. A különböző didaktika-könyvek több kérdésben eltérően tagolják e folyamatot, de az alábbi négy alapvető mozzanat nem lehet vitás:

- ismeretszerzés
- ismeretek alkalmazása
- rendszerezés, rögzítés
- ellenőrzés

Az iskolai oktatás döntő részét az ismeretszerzés és alkalmazás teszi ki, tantárgyanként eltérő arányban: matematikában, magyar nyelvtanban jelentősebb az alkalmazás ,míg történelemben jóval nagyobb az ismeretszerzés aránya. Természetesen az alkalmazásba bele is érthetünk bizonyos rendszerezést, rögzítést, ellenőrzést, ezért érthető az a felfogás, mely szerint a tanítási-tanulási folyamat az ismeretszerzés és alkalmazás ciklikus váltakozása.

Az ismeretszerzés meglehetősen összetett folyamat, a didaktika az oktatási folyamat komplex fázisaként több részmozzanatát tárgyalja:

- a tanulás pszichikai feltételeit biztosító motiváció
- az új ismeretek tényanyagának nyújtása
- a tényanyag elemzése
- absztrakciók megértése, megfogalmazása
- elsődleges rendszerezés és rögzítés
- visszacsatolások, az ellenőrzés bizonyos indirekt formái

Hol lehet, és hol érdemes a mikroszámítógép alkalmazásával próbálkozni az ismeretszerzés egyes mozzanataiban? A motivációban biztosan segít, de ez elválaszthatatlan a további mozzanatoktól. Motiválhat a következő tanári bejelentés: "Most egy szemmel nem látható jelenség számítógépes szimulációját fogjuk látni, a körülményeket Ti választhatjátok meg." Az ismeretszerzés folyamatában a szimulációt tartjuk a mikroszámítógép elsődleges alkalmazási területének, ezért indokolt kissé részletesebben foglalkozni vele.

A szimuláció kifejezés magyarul utánzást jelent. A szimuláció, mint kutatási, vizsgálati módszer egy valóságos rendszer elemeinek és az azok közötti viszonyoknak egy helyettesítő rendszerre való leképezését, és a helyettesítő rendszer segítségével való tanulmányozását jelenti. Vajon szimulációnak tekinthetjük-e egy szobor megalkotását, hiszen az is megfelel az előbbi definíciónak ? Amennyiben a szobor például anatómiai tanulmányok részére készült, akkor jogosan merülhet fel a kérdés - de mint képzőművészeti alkotás, nem a jelenség vizsgálata, tanulmányozása céljából készült. Általánosan elfogadott az a meghatározás is, amely csak dinamikus rendszerek leképezését tekinti szimulációnak, így a szobrok vonatkozásában csak a mobil szobrok jöhetnek szóba - a képzőművészet azonban nem a szigorúan precíz tükrözésre törekszik: ezért művészet!

A modell a valóság bizonyos szempontok szerint leegyszerűsített mása, egy tudományos elmélet speciális megjelenési formája (pl. atommodellek). A modell szerkezeti pontosságra törekszik, a szimuláció viszont magában foglalja az eredeti rendszerben lejátszódó folyamatoknak a tükrözését is, melyek többször megismételhetők, lejátszha-

tók. Ha tehát számítógépünkkel a képernyőre rajzoltatjuk egy molekula, vagy kristály szerkezetét – ez csak modell. Szimuláció akkor lesz belőle, ha a molekula létrejöttét, vagy a kristály feloldódását mutatja be, megismételhetően.

A számítógépes szimuláció az utóbbi évtizedek egyik legjelentősebb új tudományos módszere. Igaz ez az oktatásra is: az olcsó, minden iskolában megtalálható mikroszámítógépekkel olyan szimulációkat lehet megvalósítani, melyek új perspektívát nyitnak az ismeretszerzés számára.

Az ismeretközlés fázisában a számítógépes szimuláció lehetővé teszi más módon be nem mutatható jelenségek tanulmányozását. Ilyen terület a mikrovilág történései: elemi részecskék, atomok, molekulák mozgásai; oldódás, halmazállapotváltozások, stb. Ezekben a szimulációkban más módon nem biztosítható többletet jelent a különböző paraméterek, körülmények felhasználó általi megválasztása. Egy jelenség szimulációját változó feltételek között látva komplex, a realitásokat tükröző ismereteket nyújthatunk – egyszerűen és gyorsan. A közgazdasági képzésben elterjedt gazdasági, vállalati szimulációs programok egy bonyolult, sokparaméteres valódi környezetet szimulálva pillanatok alatt meg tudják mutatni egy konkrét döntés következményeit – számítógép nélkül hosszas számolást, elemzést igényelne ugyanez.

Nem mindent érdemes azonban számítógéppel szimulálni. A konkrét probléma elemzésével kell eldönteni, vajon nyújt-e olyan többletet, amely megéri a program elkészítését. Ismeretes olyan kémiai program, melyben két klóratom (vegyjel+külső elektronok pontokkal jelölve) közelít egymás felé, majd megállnak egymás mellett, mutatva ezzel a klórmolekula létrejöttét:

```
..          ..              .. ..
:Cl.        .Cl:    ->      :Cl:Cl:
..          ..              .. ..
```

Ugyanezt a folyamatot be lehet mutatni mágneses táblán modellekkel – semmivel nem nyújt többet a számítógépen a fenti megoldás. A motiváció természetesen erősebb lehet, de érdemes-e "verébre ágyúval lőni"? Ha ügyes grafikával az atomok, elektronok állandó mozgását is szimuláljuk, akkor már lehet, hogy megéri.

Módszertani szempontból a szimuláció kétféleképpen szolgálhatja a tényanyagnyújtást. Amikor a szimuláció alapját képező jelenség ismert a tanulók előtt, akkor a feltételek változtatásával a rendszer viselkedését lehet alapos vizsgálat tárgyává tenni, ezáltal mélyebb ismeretekhez jutni, törvényszerűségeket megállapítani – ez a deduktív megismerési mód a jelenségek analizisét teszi lehetővé. Az induktív, felfedező módszer "új" jelenségek megismerését, összefüggések feltárását biztosítja, a tanulót bevonva a kutatásba, felfedezésbe.

A számítógépes szimuláció tanításbeli alkalmazására sok lehetőséget találhatunk. Csupán kedvcsinálásként említünk néhány kézenfekvő területet:

- matematika: véletlen események, valószínűségszámítás, függvények, szimmetria
- fizika: mozgások, mikrojelenségek, elektromosság, áramlások, statisztikus mechanika
- földrajz: csillagászat, tájékozódás, áramlatok, hegységek keletkezése, térképészet
- biológia: genetika, populációdinamika, evolúció, életfolyamatok, sejt, szervek működése
- kémia: kötések létrejötte, kinetikus gázmodell, oldódás, elektrokémia, technológiai folyamatok
- technika: anyagtulajdonságok, megmunkálási folyamatok, robotműködés

- környezetismeret: természeti folyamatok, fűtési rendszerek, közlekedés, környezetünk rendezése
- történelem: híres csaták, "mi lett volna ha..." játékok
- irodalom: dráma-rendezés
- rajz-művészettörténet: szindinamika, perspektíva, fényhatások
- ének-zene: hangszer-szimulációk, zeneszerzés
- testnevelés: sportjátékok, taktika, játszmaelemzés
- idegen nyelvek: mondatszerkezetek variációi
- anyanyelv: hibaszimulációk, elemzések(helyesírás)

Az oktatási folyamat másik nagy komplex fázisa az ismeretek alkalmazása. Minden új ismeret megértése a korábbi tudáson, tapasztalaton alapul, vagyis minden új ismeret elsajátítása egyúttal a megelőző ismeretek alkalmazása is. Az utóbbi években a társadalom igénye az alkalmazás súlyát kívánja növelni: sürgetik a közoktatást, hogy inkább kevesebb, de jobban elsajátított tananyagot tanítson, növelje a gyakorlásra fordított időt az ismeretközlés rovására.

Az alkalmazás fogalmát többféleképpen értelmezik. Alapvető különbség látszik a készségfejlesztést szolgáló állandó gyakorlás és az alkotó alkalmazás között, pedig az utóbbi lehetőségét a megelőző gyakorlás adja. Egy leegyszerűsítő példa: hogy valaki jó regényt írjon, először gyakorolnia kellett a helyesírást, a fogalmazást. A tevékenység célját illetően azonban nagy a különbség: a gyakorlással - azaz reproduktív alkalmazással - egy megtanult ismeretet mélyítünk, készséget fejlesztünk, míg az alkotó - produktív - alkalmazással újat akarunk létrehozni. Az alkalmazás e két formája nem választható szét mereven: matematikában, vagy magyar nyelvtanban megszokott, hogy egy-egy szabály gyakorlása során elvileg hasonló, de valamiban új feladatot kell megoldani, s ez tulajdonképpen új ismerethez, a megoldás koncepciójának kialakításához vezethet.

Az alkalmazás didaktikai mozzanata az, amelyben minden tantárgynak sok lehetősége van a mikroszámítógépek felhasználására. Gyakorlóórán néhány számítógéppel differenciált csoportfoglalkozást lehet megvalósítani; a megfelelő program biztosítja a differenciálást. A számítógép türelme végtelen, ezért különösen alkalmas a kissé monoton gyakoroltatásra (drill and praktice): idegen szavak, évszámok, földrajzi nevek, kémiai képletek, biológiai rendszertan, helyesírás, zenei hangközök, stb. Van rá példa, hogy szünetekben a folyosón lévő gépeken ilyen programokkal gyakorolhatnak a gyerekek [21]. Fizika, kémia, biológia, technika tárgyakban a mikroszámítógépet laboratóriumi eszközként is használhatjuk. Alkalmazhatjuk egy már ismert jelenség olyan szimulációjára, mellyel a tanulók méréseket végezhetnek: mozgások, elektromos mérések, anyagcsere, reakciósebesség vizsgálata, robotműködés, stb.

Az oktatási folyamat lényeges mozzanata a rendszerezés és rögzítés. A mikroszámítógépek alkalmazása szempontjából a rögzítés területén van nagyobb lehetőség, hiszen a rögzítés legjelentősebb eszköze az ismeretek gyakorlati alkalmazása. A rendszerezésbeli felhasználásról kevés a tapasztalat, pedig sokszor grafikus módszerek (táblázat, grafikon) segítik az ismétlő-rendszerező órán a pedagógust. Vannak jó tapasztalatok az ismétlés, rendszerezés individualizálásáról, illetve csoportmunkában végzéséről. Ezek arra mutatnak, hogy készíthetünk olyan számítógépprogramokat is melyek hatékonyak segítik az egyéni, vagy csoportos ismétlést, rendszerezést.

Az ellenőrzés különböző változatai az oktatási folyamat egészében jelen vannak, de időközönként önálló didaktikai feladatként kell megjelenniük. A mikroszámítógépek alkalmazása magával hozza, hogy az ellenőrzés - és önellenőrzés - sokkal jobban átszőhesse akár az új ismeretek elsajátítását, akár a gyakorlást, vagy az ismétlést,

rendszerezést. Az első fejezetben láttuk, hogy egy oktatóprogramnak mindig jellemzője a kérdés és a válasz - legyen akár gyakorló, akár ismeretközlő. A jól szerkesztett számítógépprogram igen aprólékos és differenciált ellenőrzést, értékelést valósíthat meg, miközben a programmal dolgozó tanuló esetleg nem is gondol erre, hiszen ő gyakorol, vagy tanul. Éppen ezért kell fokozottan ügyelni a minősítéssel kapcsolatos megfogalmazásokra, hogy az támogassa a kedvező motivációt.

Az ellenörzést, mint fő didaktikai feladatot megvalósító számítógépprogramok alkalmazásával kapcsolatban sokan fenntartásokat hangoztatnak. A gyerekek könnyen tévedhetnek a gép kezelésében - s ez nem számíthat szaktárgyi hibának. Jó program ezt a tévedési lehetőséget azonban minimumra csökkentheti. A módszer jelentős pozitívuma a tanuló-gép kapcsolat feszültségmentessége, a gép objektivitásába vetett bizalom.

A felsőoktatásban gyakori a számítógépes vizsga, teszt - az eredmények hasznosnak mutatják a módszert. Az ilyen alkalmazhatóság azonban erősen életkorfüggő: minél fiatalabb diákokról van szó, annál többet jelent a tanár szerepe az ellenőrzésben - pozitív és negatív irányban is .

Vannak forgalomban olyan programok, melyek valamilyen ismeret tárgyalása után ellenőrzésre, önellenőrzésre, gyakorlásra is alkalmas feladatokat adnak, a program demonstrációt szolgáló részéhez hasonló technikával. Egy matematikai példa a "Tengelyes tükrözés" című programból[Novotrade]: a bevonalkázott képernyőn kirajzol egy alakzatot, s ezt kell a tanulónak tükröznie rajzoló és törlő funkciót ellátó gombok segítségével.Egy kémiai példa: a "Hidratáció" című programból [VORKER]: kémiai részecskéket mozgatva a kurzormozgató billentyűvel létre kell hozni megadott kémiai szerkezeteket. Mindkét programban gombnyomással be lehet kérni a feladat megoldását segítség, illetve önellenőrzés céljából. Véleményünk szerint az ilyen jellegű, főleg önellenőrzés a legcélszerűbb ellenőrzési funkciója egy programnak.

Az oktatási folyamat egyes mozzanatainak megvalósítása nem csupán a tanítási órák feladata, sok óránkivüli tevékenységnek van jelentős lehetősége a személyiségfejlesztésben: szakkörök, házi feladatok, korrepetálás, kirándulás, üzemlátogatás, egyéni pályázatok kidolgozása, stb.

A szakkörök célja a tantervi törzsanyagon túlmutató érdeklődés kielégitése, a képességfejlesztés. A természettudományos tárgyak legfontosabb óránkivüli lehetősége a szakkör, mert olyan kisérletek bemutatására, mérések elvégzésére van lehetőség, amelyek fontosak, érdekesek, de a tanítási óra időkeretébe nem férnek be. Mindebből érezhető, hogy az érdekesebb szimulációs programok, vagy szaktárgyi játék- és versenyprogramok szakkörön használhatók a legjobban : itt van idő az alaposabb kipróbálásra, megbeszélésre. Érdemes tehát szakkörön a tanítási órákon használt programokat is elővenni.

Egyes tárgyakban (főleg matematika) a házi feladatoknak kiemelkedő jelentőségük van a tanultak gyakorlásában. Reális lehetőség ma már az otthoni számítógépekre is épiteni : tizezres nagyságrendű a magántulajdonú gépek száma. Az iskola és a programkészitők fontos feladata, hogy ezt a tanulás szempontjából "holt tőkét" mozgásba hozzák: ne csak játékprogramok fussanak e sok gépen. A szülők szivesen meg is vásárolják a programkazettákat, ha az iskola felhivja rá figyelmüket. A tanulókat motiválja a számítógép, otthon szivesebben ülnek a gép, mint a tankönyv mellé.

Az iskolai tevékenységek sorában jelentős - és sajnos növekvő - szerepet játszik a korrepetálás. Itt különösen fontos az aprólékos magyarázat, többszöri ismétlés, türelem. E követelmények miatt a mikroszámítógépek iskolai alkalmazási területei között egyik

legfontosabb a korrepetálás. Néhány géppel ellátott tanteremben, szertárban, egyéb helyiségben tanítás után akár szorosan vett tanári felügyelet nélkül is gyakorolhatnak a gyerekek. Az eredményes korrepetálás olyan programokat igényel, amelyek szorosan alkalmazkodnak a használt tankönyv szövegéhez, sokirányban elágazóak, minden tanulói választ azonnal korrigálnak, és a munkát részletesen értékelik. Az ilyen programok készítése alapos elemző munkát igényel, mert ki kell térnie minden elképzelhető tanulói tévedésre, azokat részletesen meg kell magyaráznia, biztosítania kell egyes lépések hiányt nem okozó átugrási lehetőségét.

A felzárkóztatásnak nevezett kompenzáló tevékenység is eredményesen segíthető a korrepetálás céljára készült prorramokkall. Felzárkóztató tevékenységre a jobb tanulóknak is szükségük lehet betegség, vagy egyéb tanulmányi elmaradások esetén. Itt különösen fontos a programok többirányú elágaztatása, a jobb és gyengébb képességű tanulókhoz való alkalmazkodás céljából.

A Művelődési Minisztérium által kiadott szaktárgyi módszertani folyóiratok (....Tanítása) szinte mindegyike közölt már cikket a mikroszámítógépek saját tantárgybeli alkalmazásáról, legtöbbet talán a Fizika Tanítása. Mielőtt saját programot tervezne valaki, célszerű e lapokból áttekinteni a rokon tantárgyak tapasztalatait. Ehhez kívánunk segítséget adni a könyv végén közölt ajánló bibliográfiával.

Összegzésként elmondható, hogy az oktatási folyamat minden mozzanatában, minden tantárgyban találhatunk olyan témát, amelyben lehetőség van a számítógépes támogatásra. E megállapítás megfordítása azonban túlzás: mindenképpen kerülendő a számítógép állandó alkalmazása, a téma elemzését mellőző "minden áron való" programkészítés és alkalmazás.

4. Készítsünk saját programot is!

Az iskolai oktatás története meggyőzően bizonyította, hogy a minden tanulónak ugyanazt és ugyanúgy tanító iskola eredményessége egyenlőtlen. Amikor a különböző tanulórétegeknek eltérő követelményű iskolákat állítottak fel, ekkor sem lett egyenletesebb az eredmény, sőt ez a megoldás csökkentette a társadalom vertikális mobilitásának lehetőségét.

Mai közoktatásunk lépéseket tesz az individualizálás felé: választható tankönyvek jelennek meg, a könyvekben élesen elkülönül a minimum, a törzsanyag és a kiegészítő anyag - mindez új lehetőségeket ad a pedagógus kezébe. Az iskolák között nagyok a különbségek tanulói anyagban, felszereltségben, kialakult szokásokban, szakos ellátottságban, stb. A pedagógusok is individumok, nem egyformák - és nem is lehetnek azok - tanítási módszereik. Mindezek alátámasztják az egyéni, saját készítésű számítógépprogramok létjogosultságát és szükségességét.

Magyarországon jó néhány cég (túl sok is) foglalkozik szoftverforgalmazással, iskolai programokat viszont alig néhány terjeszt (pl: Tudományszervezési és Informatikai Intézet, Novotrade, VORKER). Ezek a készen vásárolható programok sok jó tulajdonsággal rendelkeznek, de vannak hátrányaik is.

A készen beszerezhető programok legnagyobb előnye az, hogy csak meg kell venni és használni - ez nem kis dolog a pedagógusok mai elfoglaltsága mellett. Az ilyen programok szerzői általában jól ismerik és maximálisan kihasználják az adott tipusú számítógép lehetőségeit, a fogyatékosságokat pedig igyekeznek sokféle programozói fogással enyhiteni. A közforgalmú programok általában szaktárgyilag és programozástechnikailag egzakt, ellenőrzött, lektorált, és szerzői jogi védelemben részesülő szellemi termékek - ezért értelemszerűen tilos a másolásuk és továbbadásuk. Szerzőik a programba védelmet is épitenek be, ezért listázásuk és SAVE paranccsal való kimentésük nem lehetséges. "Profi" programozók azonban gyakran megoldják ezt a feladatot is - igy terjednek kézről kézre a játékprogramok.

Tartalmi szempontból e programok többnyire átfogják a téma egészét, minden részletre kiterjednek. Forgalomba hozásuk előtt rendszerint iskolákban kipróbálásra kerülnek, a tapasztalatok alapján módositják szerzőik.

A készen vásárolt programok hátrányai között jelentkeznek olyan tényezők, melyeket az előnyök között is emlitettünk: nagy témákat fognak át, ezért általában hosszúak, használatuk időigényes. E programok célja, hogy mindenütt használhatók legyenek, ezért fejlesztésük során a "statisztikai átlaghoz" igyekeznek igazitani, igy nem biztos, hogy a konkrét iskola tanulóihoz, a tanár egyéniségéhez, az iskola stilusához illeszkednek. Mivel e programok védettek, a pedagógusnak nincs lehetősége a változtatásra, az igényei szerinti módositásra. Mindehhez hozzájárul egy sajnálatos technikai probléma: a számitógépekhez kapott kazettás magnók (Commodore) nagyon silány kivitelűek, gyakran elállitódnak, és igy már nem lehet beolvasni a kész programokat.

Sok készen vásárolt program legnagyobb hibája, hogy nincs világos didaktikai célja, nincs mellékelve hozzá módszertani ajánlás, útmutató. Vannak olyan programok, melyek egy tanitási egység teljes anyagát feldolgozzák -igy alkalmasak lennének tanitásra egy harminc géppel ellátott tanteremben, kissé rövidítve megfelelnének otthoni tanulásra, korrepetálásra. A pedagógusok jelentős része most találkozik először számitógéppel, az alkalmazás didaktikája nem kidolgozott, ezért felhasználásuk eredményessége igen

bizonytalan. Alapvető követelmény egy oktató célú programmal szemben, hogy világos, egyértelmű didaktikai célja legyen, s a felhasználó pontosan ismerje a programiró szándékait, javaslatait.

A saját készitésű program sok említett hátrányt kiküszöbölhet. Egy oktatóprogramot megirni BASIC nyelven nem ördöngősség, nem igényel semmiféle programozói előképzettséget. A gépek leirásában szereplő rövid BASIC-összefoglaló ismerete bőségesen elegendő programiráshoz. Természetesen egy program megirása jelentős időt igényel, és még a legegyszerűbb BASIC utasitások esetén is vannak buktatók, sok apró fogásra kell rájönni. E könyv további része ebben próbál segiteni, olyan ötleteket, mintákat adni a programok egyes elemeihez, melyekkel az említett buktatók elkerülhetők. Sokak kezdeti félelmét már itt kell eloszlatni : egy BASIC nyelvű oktatóprogram megirásához még a legalapvetőbb matematikai ismeretekre sincs szükség - a könyv végigolvasása ezt fogja bizonyitani.

A saját készitésű oktatóprogram egyéni stilusú, a megszokott tanitási módszerekhez alkalmazkodó, bármikor változtatható, fejleszthető és az iskola, osztály szintjéhez illeszthető. Tapasztalataink szerint az ilyen programok mindig rövidek, egy-egy konkrét célt szolgálnak. Technikai előny is mutatkozik : az elállitódott magnók a rajtuk felvett programot vissza tudják olvasni.

A programkészitő tevékenység hatással van a tanári személyiségre: az alkotói sikerélmény nagy motiváló hatású. A programirást megelőző elemző tevékenység a tananyag mélyebb összefüggéseit tárhatja fel, hatással van a tanitás egyéb területeire, más módszereire - a programozott oktatás tapasztalatai között már láttuk ezt a hatást.

Saját program készitése hasznos a tanárnak és a diáknak, pozitiv hatást gyakorol az oktatásra és a tanári személyiség fejlődésére egyaránt.

5. Hogyan fogjunk hozzá?

Amikor egy programot tervezünk, az alábbi kérdéseket tegyük fel először: Megéri-e ? Többet ad-e, hatékonyabb lehet-e más módszereknél ? Természetesen nem biztos, hogy ezekre a kérdésekre azonnal pontos választ lehet adni. A gondos elemzés, a lehetőségek és körülmények felmérése azonban nagy valószínűséggel csökkenti a felesleges munka veszélyét.

Egy kémiai program írását megelőző rövid elemzés talán jól illusztrálja az előbbieket. A titrálás, vagy térfogatos elemzés egyszerű kémiai analitikai módszer, az általános iskolában is tanított sav-bázis reakciók alkalmazásaként fogható fel. Olyan programot terveztünk, amely képernyőre rajzolja a büretát és a poharat, s gombnyomással lehet a mérőoldatot a büretából a pohárban lévő ismeretlen oldathoz adagolni, színátcsapás jelzi a reakció teljes végbemenését, a titrálás végpontját. Csaknem minden iskola szertárában van azonban legalább egy büretta, titrálni lehet a mindenütt meglévő nátrium-hidroxid-oldattal és sósavval. Ebből arra következtethetünk, hogy nem éri meg: helyesebb valóságban bemutatni órán a titrálást, nem igényel sok előkészületet. Felmerül viszont a tanulói kisérletezés, verseny lehetősége: a titrálás alapesete a tanulók számára egyszerű, érthető és veszélytelen. Több büretta azonban nem szokott lenni a szertárakban, de ha lenne is, a több ismeretlen oldat elkészítése olyan időigényes, hogy ezt kevés pedagógus tudná megoldani, legfeljebb szakkörön néhány tanuló számára. Úgy tűnik tehát, hogy versenylehetőséget biztosító titrálási programot érdemes készíteni: mérje a titrálás pontosságát és gyorsaságát, véletlenszerűen adja az ismeretlen oldatokat. A kémia iránti csökkenő érdeklődés miatt minden motivációs lehetőséget ki kell használni, ezért is készítettük el a programot. Az elemzésből körvonalazódik az alkalmazás legcélszerűbb módja: órán bemutatni a titrálást valóságos eszközökkel, majd a programmal szimulálva néhány tanulónak jutalom-versenyt biztosítani; órán kivüli keretben pedig más szaktárgyi versenyjátékkal kombinált bajnokságot rendezni egyének, csapatok, osztályok között.

Ha az előzetes elemzés alapján a program elkészítése mellett döntünk, akkor a program didaktikai célját kell pontosítani: az oktatási folyamat melyik mozzanatát, vagy mozzanatait kívánjuk segíteni. A fő didaktikai cél alapján az alábbi csoportosítás lehetséges:

- Új ismeret közlő program: tanithat egy konkrét tanítási egységet, vagy szimulálhat egy tanitandó jelenséget, amivel a tényanyagnyújtást valósítja meg.
- Alkalmazást szolgáló program: kérdések feladatok sorozatával gyakoroltathat, vagy laboratóriumi kisérlet, mérés vezérlésével, szimulációjával szolgálja az ismeretek alkalmazását.
- Rendszerezést, rögzítést szolgáló program: megfelelően csoportosított feladatokkal, kérdésekkel, táblázatok, grafikonok készítéséve!, bemutatásával rendszerez, rögzit.
- Ellenörző program: fő didaktikai feladatnak ellenőrzést választani meggondolandó (ld. 21. oldal).

Eddigi megfontolásaink alapján az iskolai mikroszámitógépeken megvalósítható programokat célszerűnek látszik a fő didaktikai feladat és a programozási stratégia, programszerkesztés módja szerint az alábbi csoportokba sorolni:

1. Szimulációs programok
2. Egyéni tanulást, korrepetálást segitő és új ismeretet tanitó programok.
3. Gyakoroltató programok.
4. Műszeres méréseket, gépeket vezérlő programok

A továbbiakban az első három csoportba sorolható programok készítésével foglalkozunk. A műszerek adatait feldolgozó, technikai eszközök működését vezérlő programok speciális területet jelentenek, működtetésükhöz hardver kiegészítők (interface) szükségesek, meghaladják e könyv lehetőségeit és szándékait.

Szimulációs programot akkor célszerű tervezni, ha

- a kérdéses jelenség valóságosan nem mutatható be (veszélyes, bonyolult, stb.),
- időben, vagy térben transzformálni kell, mert túl lassú, vagy túl gyors lefolyású, illetve túl nagy, vagy túl kis méretben játszódik le (csillagászat, mikrovilág jelenségei),
- a jelenség közvetlenül nem észlelhető absztrakció (történelmi, társadalmi folyamatok),
- egy valóságosan is bemutatható jelenségnél a jobb megfigyelés miatt szükséges a paraméterek szabad változtatása, és ez iskolai körülmények között nem megoldható (természettudományos kisérletek),
- jelentősen növelheti a motivációt.

Szimulációs program tervezésekor először a tanitási anyagot kell elemezni, pontosan megállapitani, hogy a programmal milyen ismeret, fogalom, tulajdonság tanitását szeretnénk elősegiteni. Ismerni kell a pontos célt és követelményeket (tanterv). Hasznos lehet a tanitási tapasztalatok összegyűjtése: mit értettek meg legnehezebben a tanulók eddig.

A szimulálandó jelenséget elemezni kell szaktudományi szempontból. Modellalkotásnál el kell dönteni, mely szempont(ok) szerint kell pontosan tükröznie a valóságot modellünknek, mely szemponttól lehet eltekinteni az adott feladatnál. A szimuláció mindig dinamikus, tehát a (statikus) modellünkkel végzett, bemutatott folyamat a tulajdonképpeni szimuláció. Mindez talán bonyolultnak tűnik, de a modellalkotás köztudottan a szaktudományok feladata. A tanitás számára készen állnak az elméleti (absztrakt, tudományos) modellek, sőt általában a konkrét fizikai modellek is, melyeket a tanitásban alkalmazni szokás (növényi részek,gépek,kristályok,hegységek,stb. modelljei). A programkészitőnek érdemes ezeket a kész modelleket tanulmányozni mielőtt munkához lát.

Az új ismeretet tanitó és egyéni tanulást szolgáló, korrepetáló programok szerkezete és funkciói sok mindenben hasonlitanak a hagyományos oktatóprogramokéhoz (ld. 2. fejezet), ezért felhasználhatjuk a kialakult programkészitési módszerek elemeit.

Az oktatóprogramok legtöbbje vagy empirikus, tapasztalati úton, vagy valamilyen pszichológiai tanuláselmélet alapján készült. A hatalmas kiterjedésű szakirodalom többnyire egyetért a programkészités fő feladataiban, eltérés a sorrendiségben és a hangsúlyban tapasztalható. Az általánosan elfogadott lépések:

- a megtanitandó anyag elemzése
- a tanulási cél kitűzése (célok hierarchiája)
- a kindulási szinvonal tisztázása
- az eredmény mérésének kidolgozása
- a program megtervezése, kidolgozása
- kipróbálás, módositás

A mikroszámitógépes oktatóprogramok készitésének első lépéséül a tanulási célok megállapitása javasolható. Meg kell különböztetni egy téma általános - nem biztos,

hogy mérhető célját. Ilyen lehet pl.:" erősítse a történeti szemléletet", "fejlessze a fizika törvényeinek felismerését a mindennapi történésekben", stb. Ezeket a célokat szem előtt kell tartani a programkészítés teljes folyamatában, de nem köthetjük őket egy-egy konkrét programrészhez. Minden témának megállapíthatók konkrét (mérhető) részcéljai, ebben segítenek a tantervi követelmények, tanmenetek, tanári kézikönyvek. Ilyen részcél pl.: "Tudjanak egyenlő nevezőjű törteket összeadni, kivonni","ismerjék fel és nevezzék meg az összetett szavak szófaját". Érdemes e részcélokat még jobban lebontani - ami egyúttal már a tananyag elemzését, blokkokra bontását jelenti, pl: "tudják a szerves savak funkciós csoportjának nevét, képletét". E blokkok megfelelő rendszerbe állítása, a köztük lévő kapcsolatok jelölése lényegében az oktatóprogram folyamatábráját adja.

A célrendszer megállapítása mellett a kiinduló helyzet tisztázása is szükséges: melyek azok a meglévő ismeretek, amelyekre építhetünk a program során. Ezt egyrészt ismernünk kell elméletben (tanterv szerint), de érdemes gyakorlatilag is (teszt) felmérni, ez alapján lehet a programba beépíteni a hiányos előismeretek pótlását biztosító elágazási pontokat, kerülő utakat.

Lineáris programok készítésére fejlesztették ki az Amerikából elterjedt RULEG-módszert [22]. A tisztán lineáris programokat nem tartjuk követhetőknek, de a módszer elemei jól használhatók a számítógépes oktatóprogramok előkészítésekor is. A részcélokat tükröző blokkok többnyire lineárisan helyezkednek el, a blokkokon belül kell minél több elágazással biztosítani a az optimális haladási irányt. A RULEG-módszerből kiindulva követhetjük az alábbi műveletsort:

1. Tanulási cél kitűzése.
2. Az anyaghoz tartozó összes szabály meghatározása, fogalomjegyzék összeállítása, a fogalmak lebontása elemi szabályokra, megállapításokra.
3. Az elemi szabályok logikus sorrendbe állítása, a programszekvencia megalkotása.
4. A programszekvenciába rendezett szabályok sorrendjének felülvizsgálata és módosítása a közöttük lévő kapcsolatok alapján.
5. Folyamat-diagram összeállítása a módosított szabálysor szerint: visszalépések, kerülő utak kijelölése.
7. A szabálysor elemeinek program-modulokká alakítása (ld. következő fejezet).

A logikus szabálysor megállapítása, a módosítás gyakran az illető tananyagrész átstrukturálását jelenti. Ezzel a módszerrel tehát programkészítéstől függetlenül is hasznosítható elemzéseket végezhetünk egy-egy téma, tankönyvrészlet szerkezetén.

A gyakoroltató programok könyvünk szempontjából talán a legfontosabbak, mert az iskolai gépállomány (néhány gép iskolánként) és az otthoni gépek esetleges alkalmazása leginkább az egyéni gyakorlást támogatja, és főleg mert e programok készítése sokkal kevesebb programozói ismeretet kíván, mint a - többnyire grafikus - szimulációs programoké.

A gyakorlás lényege a megértett, emlékezetben tárolt ismeret céltudatos alkalmazása, felhasználása, aktualizálása. A gyakorlásban alapvető a reproduktív alkalmazás, ezzel érhetjük el a megfelelő jártasságok, készségek kialakítását, amelyek alapjai a produktív, alkotó alkalmazásnak. A gyakorlás alapvető feltétele a tudatosság, tehát a gyakorolni kívánt ismeret tökéletes megértése. Sokszor a gyakorlás célja nem a tanult ismeretek alkalmazása, hanem az ismeretek jobb megértése, bevésése.

A gyakoroltató programok készítésének első lépése a pontos,mérhető cél kitűzése kell legyen: mely fogalmakat, ismeretet, módszert akarjuk gyakoroltatni, és milyen szintre

kívánjuk eljuttatni a tanulókat a programmal. A mérhető cél kifejezést itt lehet szó szerint is érteni, pl. "30 másodperc alatt állapitsa meg az egyszerű mondat alanyát és állitmányát"(az idő mérését a program biztosithatja). Ki kell gyűjteni mindazt a fogalmat, ismeretet, amelyekre épithetünk, de a gyakorló programnak bármely ismeret hiányát tudnia kell pótolni. A program elágaztatását a tanuló válasza, helyenként segitségkérése döntse el.

A tananyag elemzését gyakoroltató programnál a lehetséges tévedések, hibák, az ezzel kapcsolatos tapasztalatok szempontjából kell végezni, arra törekedve, hogy ne lehessen olyan tanulói tévedés, melyre a program nem tud konkrét választ adni.

A gyakorlotató programok legjobban igénylik a "karbantartást", azaz a tapasztalatok alapján történő állandó továbbfejlesztést, módositást. A legnagyobb tapasztalattal rendelkező pedagógus is kaphat programjában olyan tanulói választ, melyre az elemzésnél nem gondolt. A program bővitése ilyen válaszvariánsokkal technikailag igen egyszerű a saját programnál – egy készen vett (védett) programnál viszont lehetetlen.

Az iskolákban elterjedt mikroszámitógép-tipusok között vannak olyan különbségek, melyek befolyásolhatják a tervezett program célszerűségét. Egy program-terv alapján ki lehet választani a legalkalmasabb géptipust – amennyiben van választékunk:

- Képleteket, hatványokat, indexszámokat tartalmazó szöveges programot célszerű PRIMO gépen kivitelezni.
- Rajzokat, ábrákat (szineseket is) igénylő programokhoz jobb a Commodore 16, plus/4 és TVC.
- Mozgó figurákat (sprite) használó programot Commodore 64 gépen lehet könnyebben és jobban kivitelezni.
- Speciális karaktereket (pl. gyökjel, görög betűk, stb.) igénylő programot TVC gépen valósithatunk meg könnyen.

E megállapitások elsősorban a BASIC nyelvű programozásra – kezdőkre – vonatkoznak. A gyakorlott programozók gépi kódu programokkal, rutinokkal át tudják hidalni a gépek közötti különbségeket. Mint látni fogjuk a későbbiekben egyszerű BASIC programokkal is van lehetőség bármelyik gépen megoldani feladatunkat, legfeljebb minőségi különbségek lesznek a géptől függően.

Összegezve az eddig leirtakat, elmondhatjuk, hogy nyugodtan használható az "elfelejtett, elásott" programozott oktatás sok módszere, mert a mikroszámitógép adta meg azt a technikai lehetőséget, ami valóra válthatja a húsz év előtti "divat" előremutató elemeit, humanizálhatja a merevnek, mechanikusnak tartott egykori programozott oktatást. Bátran elővehetünk akkori oktatóprogramokat (irodalomjegyzék), tanulmányozásuk segiteni fog a számitógépprogramok tervezésében, a témák kiválasztásában.

6. Hogyan építsük fel programunkat?

Ebben a fejezetben a tanitó, gyakoroltató, korrepetáló programok felépitését boncolgatjuk, e programok ugyanis nagyon hasonló részekből állithatók össze. A szimulációs programok rendszerint egyedi megoldásokat igényelnek, melyekben a grafikus képernyő programozástechnikai problémái a döntőek - a további fejezetekben fogunk kitérni a szimulációs programokra.

Lényegében három alapmodulból felépithetjük a legtöbb programot :

- Információ (közlés)
- Kérdésfeltevés (kérdés-válasz)
- Válaszelemzés

Az információs modul megfogalmazása és kivitelezése jelenti a legkevesebb problémát. Minden közlendőt ezekbe a modulokba kell sűriteni, ezért különösen fontos a tömörités, a világos, egyszerű fogalmazás. Nem szabad a képernyőt teljesen teleirni tördelés nélkül, mert ez pszichikailag kedvezőtlen és nehezebben olvasható a gyerekek számára. Vannak olyan programok, amelyek lassan, betűnként, szavanként irják ki a szöveget - nem tartjuk jónak ezt a megoldást, mert a gyorsabban olvasót (pl.a tanár) lassitja munkájában. Egyszerűbb és célszerűbb a gép gyors alap kiiró módját használni, de megfelelő időt adni a szöveg feldolgozására. Nem szerencsés az információs modulban maximálni az elsajátitásra szánható időt. Általános tapasztalatok szerint legjobb megoldás, ha a felhasználó bármikor gombnyomással jelezheti továbbhaladási szándékát.

A képváltások gyakorisága nem jó a szemnek, ezért meg kell találni azt az optimumot, ameddig teleirhatjuk a képernyőt. Sorkihagyás, vagy más tördelés nélkül 10 sornál többet ne irjunk ki. A különböző funkciójú modulokat nem kell feltétlenül képváltással elválasztani, egy információs modul után következő kérdés természetszerűleg lehet egy képen az információval, illetve egy részével. A válasznak azonban a kérdéssel egy képen kell lennie, akár feleletválasztást, akár feleletalkotást igénylő a kérdés, ezért a válasz helyigénye döntő lehet. Mivel a képernyőn elhelyezhető karakterek száma véges (általában 24 sor, soronként 40 karakter), ezért szükség lehet a kérdés feltevésekor visszalapozási lehetőséget biztositani a felhasználónak: gombnyomással újra kérhesse a már elolvasott szöveget.

Az információs modul kivitelezésében egy technikai nehézség lehet számottevő: a képletekben az alsó-, vagy felső index. A PRIMO gépeknek alapfunkciói között ez szerepel, a Commodore, TVC és a legtöbb más gép esetében két megoldás szokásos: új karakterként definiálni alsó- és felsőindexeket (ld. következő fejezet), vagy a számokat másik sorba irni:

$$E = mc^2 \qquad H_2SO_4$$

A rossz válaszokat a programnak értékelnie kell szövegesen, rámutatva a hibára, azt megmagyarázva, esetleg visszairányitva egy korábbi információhoz, vagy új kérdést feladva dönteni el a továbbhaladást. A programnak ez a része egy kisegitő információs modulként fogható fel, amelynek tartalma a konkrét hibára vonatkozik.

A kérdésfeltevésben egyaránt szükség van a pedagógiai tapasztalatra és a számitógép lehetőségeinek figyelembe vételére. A kérdésfeltevő modul nélkülözhetetlen része a válasz megadási módjának elmagyarázása. Alapvetően kétféle válaszadási mód lehetséges: feleletválasztásos és feleletalkotó. A feleletválasztásos módszernél a tanuló látja a lehetséges válaszokat és döntését gombnyomással jelzi. Ennél a megoldásnál megkér-

dőjelezhető a tényleges szellemi erőfeszítés, hiszen a felismerés szintje elegendő a válaszadáshoz. Számítógépes oktatóprogramban sokkal kevésbé indokolható a feleletválasztásos technika, mint az írásos programozott anyagokban, egyszerű oktatógépeknél. Alkalmazását ezért minimálisra kell szorítani.

A feleltalkotást igénylő kérdések használják ki igazán a számítógép adottságait. Teljesen szabadon, kötetlenül fogalmazott, "mesélő" válaszok értékelése azonban csak bonyolult szövegelemző programokkal lehetséges, ezért jelentősen korlátoznunk kell a feleletalkotás szabadságát. A legegyszerűbb, de jól használható megoldás, amikor válaszként néhány szavas nevet, kifejezést, adatot, verssort kérünk. A programnak tartalmaznia kell az összes elképzelhető választ, s az ezekkel való összehasonlítás alapján a gép pillanat alatt képes értékelni. A szaktárgyi tapasztalat és a programozói türelem dönti el a figyelembe vett válaszlegetőségek számát – a kipróbálások itt eredményezhetnek leginkább változtatásokat. A gyerekek fantáziája azonban végtelen, ezért mindig készen kell állnia a programnak a számára érthetetlen válaszra. Bizonyos korlátozott lehetőség van egyszerű BASIC-programban is arra, hogy a válasz egyes részeit külön is megvizsgálva ne csak a teljes válasz összehasonlításával értékelhessen a program.

A kérdésfeltevő modul nélkülözhetetlen része a megkívánt válasz formai jegyeinek tökéletesen egyértelmű és világos közlése. Ki kell térni a következőkre: kisbetű, nagybetű, írásjelek, kötőszók, számok, indexek, kitevők, mértékegységek, javítási lehetőség, válaszírás befejezésének jelzése. Kiemelten kell hangsúlyozni több alkalommal is – amíg nem automatizálódik – a magyar ékezetes betűk használatának módját. A lehetséges hibás válaszok programbeli felsorolásánál sem szabad elfeledkezni az ékezetek esetleges hiányáról. Az egyszerűbb feleletválasztásos kérdések esetén is jelezni kell, hogy a lenyomott betűt, vagy számot lehet-e javítani és kell-e utána RETURN-gombot nyomni.

Vannak olyan kérdések, melyekre a választ semmiképp nem tudjuk olyan formában megtervezni, hogy a tanuló önállóan írhassa be a gépbe. Ekkor a feleletválasztásos módszert kell alkalmazni, de "enyhíthetjük" a következő módon: a válasz írásbeli megadására utasítjuk a tanulót, azzal, hogy elkészülte után nyomjon meg egy gombot, s csak ekkor látja meg az alternatívákat, melyekből ki kell válassza az általa leírtat. Az információs modullal ellentétben a kérdésfeltevésnél elképzelhetőnek tartjuk a felhasználható idő megadását, ekkor azonban állandóan ki kell jelezni láthatóan az idő múlását. Pszichikailag semmiképpen nem jó hatású, mert feszültséget kelthet, ezért inkább csak verseny jellegű feladatoknál alkalmazzuk. Az viszont javasolható, hogy egy-egy programrész, illetve az egész program futása közben eltelt időt közöljük a tanulóval, és beszámítsuk munkája értékelésébe.

A válaszelemző modul kivitelezése általában több gondot okoz az előbbieknél, feladatai: a tanulói válasz felismerése, azonosítása, hibapontok regisztrálása és a program elágaztatása. E modulnak kell tartalmaznia az előbbiekben már említett összes elképzelhető válaszlehetőséget és ezek következményeit (pontozás, továbbirányítás). A hibaregisztrálásra nagyon jó lehetőséget ad, hogy a mikrogépek sok változóval tudnak dolgozni, így gyakorlatilag egy program során elkövetett összes hibát nyilvántarthatjuk és felhasználhatjuk az értékelésben, vagy az elágaztatásoknál. A programmal való munka után e hibapontok alapján különböző szöveges értékelést kaphat a tanuló; a pedagógus pedig részletes tapasztalatokhoz jut.

A modulok összekapcsolása megtörténhet lineárisan, vagy elágazóan. A lineáris programnál az információ – kérdésfeltevés – válaszelemzés – (kisegítő információ) sorrend ismétlődik folyamatosan, ahol a kisegítő információ tartalmazza a rossz válasz korrigálását. Lineárisak lehetnek leggyakrabban a versenyt, feleltetést megvalósító, vagy a tanári demonstrációt szolgáló (többnyire szimulációs) programok. Ilyen egyszerű lineáris program pl. egy idegen nyelv tanításában gyakorlásra használt szókikérdező program. Egy magyar szó kiírása után a tanulónak be kell írni a szó idegen nyelvű megfelelőjét (vagy fordítva), a program értékeli a választ, majd adja az újabb kérdést (lista a később-

biekben). Bonyolultabb verseny-játék programok is gyakran lineárisak, pl. az előzőekben elemezett kémiai titrálási program. Sémája:

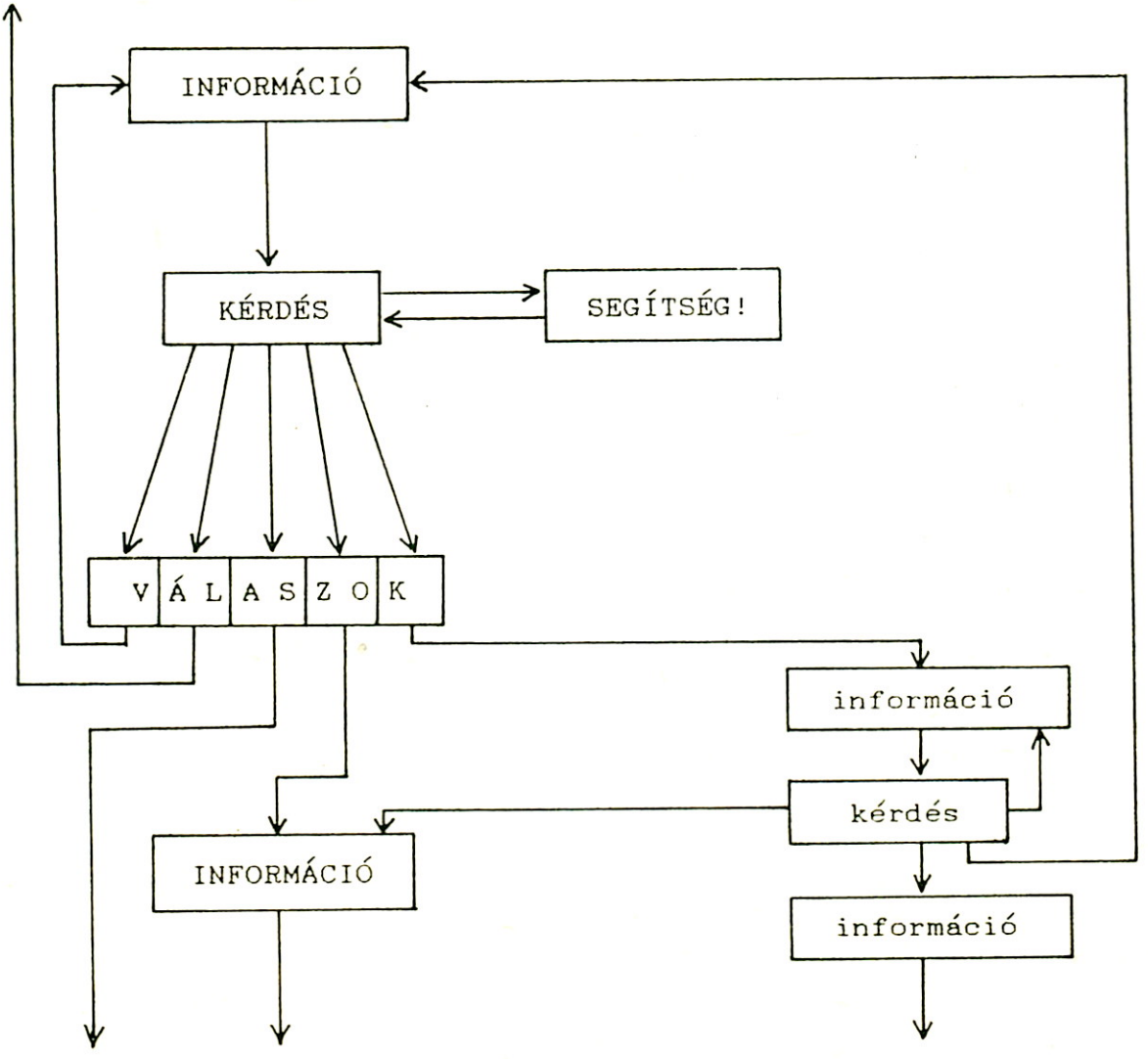
Információ -> feladatleirás -> 1.játék -> 2.játék -> 3.játék -> értékelés

Minimális elágazási lehetőséget az ilyen programnak is kell tartalmaznia, pl. a feladatleirás után érdemes megkérdezni, hogy indulhat-e a játék, vagy újra kéri a leirást, információt.

A tanitó, gyakoroltató, korrepetáló programok esetében a modulokat elágazásos programmá jobb összefűzni. Helyes tanulói válaszadás esetén lineáris programhoz hasonlóan a következő információt kapja a tanuló. Hibás válasz esetén a válasz tartalma dönti el a továbbhaladás útját, amelyre több lehetőség is van a válasz minősitése után:

- újra a kérdést feladni
- a kérdést megelőző információt adni újra,
- a program korábbi részére visszalépni,
- kisegitő információ után továbbhaladni,
- kisegitő információ utáni újabb kérdéssel eldönteni a mellékágon haladást, vagy főágon továbbhaladást.

Pedagógiailag helyes az a módszer, amikor segitségkérési lehetőséget biztositunk válaszadás előtt. Ekkor a kérdés mellett ki kell irni: "Ha .....ismeretben nem vagy biztos, ird le: segitség!(nyomd meg az S betűt)." Természetesen a hibás válaszokhoz hasonlóan a segitségkérések is pontosan regisztrálhatók egy változóval. A vázolt elágaztatási lehetőségeket érzékelteti az alábbi ábra:



7. A modulok gépi megvalósítása

Az előző fejezetben tárgyalt modulok megvalósításához tulajdonképpen a BASIC nyelv alapelemeinek ismerete elegendő. A továbbiakban feltételezzük, hogy az olvasó birtokában van ezen alapismereteknek, ezért nem tárgyaljuk a közismertebb, egyszerű parancsok, utasítások (run,load,input,print, stb.) jelentését, szintaxisát. Javasoljuk, hogy programíráskor mindig legyen kéznél a gép felhasználói kézikönyve, amelyből pillanatok alatt kikereshető egy-egy utasítás használatával kapcsolatos minden tudnivaló. Könyvünkben csak a konkrét példákkal kapcsolatos legszükségesebb BASIC-ismeretek kerülnek említésre. Mivel ezt a könyvet a programírást most kezdők számára készítettük, igy a BASIC alapjainál több ismeretet sem feltételezünk, minden megoldást a BASIC lehetőségein belül keresünk, nem térünk ki a gépi kódu programozással megoldható feladatokra. Néhány fejezet a tankönyvekből ismert "kiegészítő anyag"-ként fogható fel (karakterszerkesztés, sprite-grafika, C64 hangkeltése), ezek nehezebbek a könyv többi részénél, de kihagyásuk nem érinti az egyéb fejezetek megértését. A könyv végén lévő ajánló bibliográfiában néhány jó BASIC-tankönyvet és egyéb szakkönyvet említünk meg.

7.1. Az információs modul

7.1.1. Szöveg közlése

Az információ közlése történhet szöveggel, táblázatokkal, rajzokkal, ábrákkal - leggyakrabban ezek kombinációival. A következő fejezetben külön elemezzük a grafikai megoldásokat, ezért itt csak a szövegek és táblázatok készítésével foglalkozunk.

A szöveges információban maximálisan alkalmazkodni kell a könyvekben, egyéb nyomtatott anyagokban megszokott formákhoz. Ezért legjobbnak tartjuk a kis és nagybetűk megszokott használatát. A PRIMO és TVC gépek bekapcsoláskor ebbe, az úgynevezett irógépmódba, vagy nagybetű/kisbetű módba kerülnek, míg a Commodore gépek alapállapota a nagybetű/grafika. Commodore programok írásakor ezért a program elején be kell állítani az irógép-üzemmódot. Ennek legegyszerűbb módja a PRINT CHR$(14) utasítás kiadása. Ezután a felhasználó kézzel (SHIFT+C=) vissza tudja váltani az üzemmódot, de ha meg akarjuk tiltani, írjuk a programba: PRINT CHR$(8). Sajnos ez a tiltás az úgynevezett "Commodore-gombot" (C=) tiltja, ezért a továbbiakban az is hatástalan lesz. E tiltás feloldása: PRINT CHR$(9). Természetesen egy programon belül ezen utasításokat bármennyiszer kiadhatjuk, igy, ha fontosnak érezzük a tanulók számára a váltás letiltását, minden olyan esetben hatástalanítani kell egy-egy programrészre, amikor a felhasználónak szüksége van a C= gombra, pl.:ékezetes betűk beírása egy kérdésre. A programíró munkáját könnyíti, ha ezt a tiltást csak a program elkészülte után, a program elejére írja be.

A képernyőre írt szövegben többféle kiemelést is végezhetünk: inverz, villogó, aláhúzott, színes betűk - sajnos ezek nagyrészt gépfüggőek, nem mindegyiket lehet bármelyik gépen BASIC-programmal megvalósítani. Az említett kiemeléseket a PRINT utasításban az idézőjelen kívül írt CHR$ függvény megfelelő értékadásával érhetjük el, pl. PRINT CHR$(18)"A" a Commodore gépeken inverz A betűt eredményez: a karakter és az alap színe felcserélődik. A kiíratás szempontjából fontosabb CHR$ kódokat foglaltuk össze az alábbi táblázatban.

	C 16, plus/4	C 64	PRIMO
inverz be	18	18	4
inverz ki	146	146	20
villogás be	130	-	-
villogás ki	132	-	-
cursor le	17	17	-
cursor fel	145	145	-
cursor jobbra	29	29	32
cursor balra	157	157	8
aláhúzás be	-	-	5
aláhúzás ki	-	-	21
nyújtott karakter be	-	-	2
nyújtott karakter ki	-	-	18
alsóindex	-	-	16
felsőindex	-	-	17

Amikor egyes karaktereket külön szeretnénk színezni, hasonlóképpen adhatjuk meg a szint:

```
PRINT CHR$(158)"A"
```

egy sárga A betűt ir a Commodore gépeken. A szinek CHR$ kódját mindig a gépkönyvben kell megkeresni, mert még a Commodore egyes gépein is különbözőek. Ne felejtsük el, hogy egy ilyen vezérlő karakter hatása mindaddig megmarad, amig nem adunk ellentétes, illetve módosító értelmű vezérlő karaktert.

Az említett kiemeléseket a Commodore gépeken más módszerrel is elérhetjük: a kiirandó szövegbe a PRINT utasítás idézőjelén belül a megfelelő billentyű lenyomásával is bevihetjük a vezérlő karaktereket:

```
PRINT"H<crsr le>2<crsr fel>O"
```

A <zárójel> a benne jelzett gomb megnyomását jelenti a továbbiakban. Ilyenkor a vezérlő billentyűk lenyomásának hatására a képernyőn valamilyen inverz jel jelenik meg, de programfuttatáskor a megfelelő vezérlő utasítás hajtódik végre. Ugyanez a módszer használható a karakterek szinezésére is, ekkor az idézőjelen belül a megfelelő szinbillentyűket kell használni.

A Commodore 16 és plus/4 gépek érdekes lehetősége, hogy a képernyőn ablakot definiálhatunk, s ezután a gép csak ezt a képernyőterületet használja, a többi rész változatlan marad az ablak megszüntetéséig. A gépkönyvek az ablak parancs módban történő megadását irják le: az [ESC] billentyű és a [t] betű együttes lenyomása az ablak bal felső, mig az [ESC] billentyű és a [b] betű együttes lenyomása az ablak jobb alsó sarkát határozza meg az aktuális kurzorpozíciónál. Megfelelő vezérlő karakterek segítségével programból is egyszerűen definiálhatunk képernyőablakot: az [ESC] billentyű lenyomását a CHR$(27) kóddal helyettesithetjük. Az ablak megszüntetését az [ESC] billentyű és az [n] betű együttes lenyomása biztositja. A PRINT utasításban alkalmazott kurzormozgató vezérlő karakterekkel a képernyő bármely részén kialakithatunk ablakot.

A képernyőre iratásnál lehetőség van az irógépeknél megszokott tabulálásra is :

```
PRINT TAB(20)"Gyorsulás"
```

Ez a programsor a 20. betűhelyen kezdi kiírni a szöveget. Ugyanezt elérhetjük az idézőjelben lévő szóközökkel is, de az kényelmetlen és csúnya megoldás, ha nem csak néhány betűhelyről van szó.

Az eddig elmondottakat mutatja be az alábbi rövid programlista C 16 vagy plus/4 gépen.

```
20 print"<CLR>"
30 print"H<CRSR le>2<CRSR fel>O"
40 print
50 print tab(20)"E=m*c<CRSR fel>2<CRSR le>"
60 print"              1848"
70 print"H"chr$(17)"2"chr$(145)"O":print
80 print chr$(27)"t"
90 print"<CRSR jobbra><CRSR jobbra><CRSR jobbra><CRSR le><CRSR le>
"chr$ (27)"b"
100 print"bbbbbbbbbbbbbbbbb"
110 for i=1 to 5000: next i
120 print chr$(27)"n"
```

A programok beírásánál mindig a beállított üzemmódnak megfelelően kis-, vagy nagybetűkkel kell írni a BASIC-szavakat, a kiírandó szövegben (PRINT után idézőjelben) használhatunk tetszőleges betűket. A Commodore 16 és plus/4 gépeken a magyar ékezetes betűk miatt a kisbetű/nagybetű üzemmódot kell használni, ezért a könyvben közölt programlistákat így írjuk. Szöveg közben a kiemelés céljából írjuk nagybetűkkel a BASIC-szavakat.

A fenti listában a 20-as sor a képernyő törlését végzi, minden program elején alkalmazni kell (idézőjel után egyszerre megnyomott [SHIFT] és [CLR] gombok). A listában látható néhány "üres" PRINT utasítás egy-egy üres sort hoz létre a jobban áttekinthető képernyő érdekében. Futtatáskor a 100-as sor érzékelteti, hogy a program csak a képernyőablak területén dolgozik. A 110-es sor szerepe a lassítás, mielőtt törölnénk az ablakot.

A C16 és plus/4 gépek CHAR utasításával a képernyő bármely részén elhelyezhetünk szöveget:

```
CHAR a,x,y,"szöveg",b
```

(a) a szintipus (ld. később), ha a=1, akkor az írás színét használja. (x,y) a "szöveg"-et pozicionálja: (x) kezdő oszlopszám (0-39), (y) kezdő sorszám (0-24). (b) az írásmód, ha értéke 0, akkor normál, ha b=1 , akkor inverz a kiírás. A CHAR utasítás elsősorban a rajzok, táblázatok készítésénél válik fontossá, grafikus üzemmódban is ugyanígy használható.

A PRIMO gépek PRINT$ utasítása fentivel azonos szerepet tölt be:

```
PRINT$y,x,"szöveg"
```

(y) kezdő sorszámmal (0-15), (x) kezdő oszlopszámmal (0-41) írja ki a "szöveg"-et.

A TVC gépek hasonló utasítása a PRINT AT:

```
PRINT AT y,x,"szöveg"
```

(y) kezdő sorszámmal, (x) kezdő oszlopszámmal írja ki a "szöveg"-et.

A Commodore gépek színeket is tudnak kezelni. A karakterek színének beállítását már láttuk, de szükség lehet a keret és a háttér színének megváltoztatására - a bekapcsoláskor beálló szinek nem a legjobbak. A C16 és plus/4 gépek BASIC nyelvében külön utasítás áll rendelkezésre: COLOR a,b,c . A három paraméter közül az első (a) egy kódszám, értéke 0, ha a háttér; 1, ha a karakterek és 4, ha a keret színét kívánjuk beállítani. A második paraméter (b) a szint adja meg 1 és 16 között (ld. gépkönyv), a harmadik paraméter (c) a színerő értéke 0 és 7 között. A harmadik paramétert elhagyhatjuk, ekkor a legnagyobb színerőt (7) állítja be az utasítás.

A C64 gép BASIC nyelve nem tartalmaz ilyen utasítást, igy a POKE utasítással kell közvetlenül a megfelelő címre beírni a kívánt színeket. A címek: keret:53280, alap: 53281. Az alábbi programsor tehát kék keretben sárga alapot állít be:

```
10 POKE 53280,6:POKE 53281,7
```

A TVC gépek az egyes területek színének beállítására külön-külön BASIC utasítással rendelkeznek:

```
SET INK a        : írás szine
SET PAPER a      : háttér szine
SET BORDER a     : keret szine
```

(a) a gépkönyvben található szinkód.

Speciális esetben szükség lehet arra is, hogy a képernyő megadott pontjára kiírassunk egy karaktert, függetlenül attól, hogy egyébként mi van a képernyőn.Ezt a feladatot is a POKE utasítással végezhetjük el a képernyőmemória címeinek ismeretében. A gépkönyvek általában táblázatos formában tartalmazzák e címeket. Példaképpen: a C64 képernyőmemóriájának kezdőcíme 1024. Egy képernyősor 40 karakterhelyet tartalmaz, ezért a további sorok kezdőcímei: 1064, 1104, stb.(25 a sorok száma). Minden megjelenithető karakternek megfelel egy képernyőkód, melyet a POKE utasításban használni kell. Mivel egyszerre csak az egyik karakterkészlet használható, ezért egy képernyőkód két karaktert is jelent. Az alábbi programsor a képernyő második sorának 10. karakterhelyére ir egy [a], vagy [A] betűt a beállított üzemmódtól függően:

```
20 POKE 1074,1
```

A karakterkódokat szintén a gépkönyvekben találhatjuk

E fejezet összefoglalásaként azt tanácsoljuk, hogy programirás előtt minden itt említett és esetleg máshol látott, olvasott lehetőséget próbáljunk ki, tanulmányozzuk át alaposan a gépkönyveket - különösen a PRINT utasítás lehetőségeit.

7.1.2. A karakterek módositása

Fizikai és kémiai programok irásakor komoly gondokat okoz az indexszámok kiirása. A PRIMO gépek nagyszerű tulajdonsága, hogy alsó és felső indexeket is tudnak irni a már emlitett CHR$ kódok alkalmazásával, illetve a PRINT utasitásban idézőjel után a [CTR] és [P] gombok (alsó index), [CTR] és [Q] gombok (felső index) egyidejű lenyomásával. Commodore gépeken az indexszámok másik sorba irása egy lehetséges megoldás, de elegánsabb és didaktikailag is jobb új karaktert definiálni az indexszámok céljára. A TVC gépeken igen könnyű lehetőség van tetszőleges karakterek tervezésére.

Az eredeti karakterkészlet megváltoztatása nem könnyű feladat kezdő programozók számára. Lehet készen kapni karakterszerkesztő programokat, melyek több-kevesebb sikerrel használhatók saját programban. A szakfolyóiratok gyakran közölnek ilyen programokat (listával), melyek célja általában a magyar ékezetes betűkészlet használata az ezt nem tartalmazó gépeken, de ezeket a programokat egy kis ügyességgel fel lehet használni új karakterek szerkesztésére is. C64 géphez megfelelő ilyen programot olvashatunk a Mikroszámitógép Magazin 1986. júniusi és októberi számában, cirill betűket szerkesztő programot pedig az 1986. 11-12. számban. PRIMO gépre közölt karakterszerkesztő programot az Ötlet 1986-ban.

A C16 és plus/4 gépek elterjedtsége és a viszonylag egyszerűbb lehetőség miatt indokoltnak érezzük, hogy bemutassunk egy könnyű módszert e gépeken új karakterek szerkesztésére. Aki nem kiván ennyire belemélyedni a programozásba, nyugodtan kihagyhatja ezt a fejezetet, a továbbiak megértését nem befolyásolja. A TVC már emlitett egyszerű megoldását a fejezet végén ismertetjük.

A számitógépek memóriájának két része a ROM és a RAM. A ROM (Read Only Memory= csak olvasható memória) tartalma csak kiolvasható, nem módositható, erre a memóriaterületre nem lehet információt beirni, a ROM tartalmazza a gép működéséhez szükséges összes - gyárilag beégetett - információt. A RAM (Random Access Memory= véletlen hozzáférésű memória) gyárilag üres, erre a területre bármilyen információt beirhatunk, azt kiolvashatjuk - a gép kikapcsolásakor azonban tartalma elvész. A gépbe bármilyen programot, adatokat viszünk be, az mindig a RAM-ba kerül.

A C16 és a plus/4 gépek alapvetően csak a tár méretében különböznek: a RAM mérete a C16-ban 16K, a plus/4-ben pedig közel 64K. Az alábbi vázlatos memória térképen a C16 tára egyes részeinek funkcióit láthatjuk:

cím(dec)	cím(hex)	
0 - 2047	$0000 - $07FF	rendszerváltozók
2048 - 3071	$0800 - $0BFF	szinmemória
3072 - 4095	$0C00 - $0FFF	képernyő memória
4096 - 16383	$1000 - $3FFF	szabad BASIC-terület
32768 - 53247	$8000 - $CFFF	BASIC ROM
53248 - 55295	$D000 - $D7FF	karakter ROM
55296 - 65535	$D800 - $FFFF	KERNAL ROM

Amennyiben nagyfelbontású grafikát használunk, a szabad BASIC terület jelentősen csökken, ugyanis a grafikus képernyő bittérképe, fény- és szinmemóriája lefoglalja a 6144 - 16383 ($1800 - $3FFF) területet.

A PLUS/4 és a tárbővitővel ellátott C16 gép szabad BASIC területe (RAM):
4096 - 64768 $1000 - $FD00

Rögtön szembeötlő, hogy ez utóbbi esetben 32768 - 64768 ($8000 - $FD00) közötti címeken ROM és RAM is van. Ennek technikai okai vannak, a gépek operációs rendszere (KERNAL) biztosítja a két terület közti megfelelő "lapozást". Részletesebben tárgyalja a témakört pl. az alábbi könyv: Babán G.-Masa I.: Gépi kódú programozás kezdőknek és haladóknak. (Novotrade, 1987).

A ROM tartalmazza a gép által használható karakterek alakját is (karakter-generátor ROM). Minden karakter egy 8x8-as pontráccsal ábrázolható, a pontok mindegyike lehet bekapcsolt (1), vagy kikapcsolt (0) állapotban. A karakter teljes alakjának tárolásához 8 bytera van szükség, ahol egy-egy byte a pontrács egy-egy sorát reprezentálja. A C16 és plus/4 gépekben a karakter memória az 53248 ($D000) címtől az 57343 ($D7FF) címig tart. (nagy betű/grafikus: $D000 - $D3FF, kis/nagy betű: $D400 - $D7FF). A karakterek között második a nagy A betű:

kép	bináris	dec.	hex.	cím(dec)	cím(hex)
**	00011000	24	18	53256	D008
****	00111100	60	3C	53257	D009
** **	01100110	102	66	53258	D00A
** **	01100110	102	66	53259	D00B
******	01111110	126	7E	53260	D00C
** **	01100110	102	66	53261	D00D
** **	01100110	102	66	53262	D00E
	00000000	0	00	53263	D00F

A bináris, dec.(=decimális) és hex.(=hexadecimális) jelölések a kettes, tizes és tizenhatos számrendszerekre utalnak, melyekről valamennyit tudni kell a karaktermódosításhoz. A bináris szám két jegye (0 és 1) pontosan visszaadja pontrács ki- és bekapcsolt pontjait. Decimális számokra a BASIC-ben, hexadecimális számokra pedig a gépi kódban van szükség. A karakter tervezésekor a megrajzolt pontrács alapján könnyű az egyes byte-ok (egy-egy pontsor) értékét felírni bináris alakban : bekapcsolt=1, kikapcsolt=0. E bináris számot úgy alakíthatjuk át decimálissá, hogy jobbról balra haladva felírjuk 2 hatványait 0.-tól 8.-ig és alá írjuk a byte egyes bitjeit. Az 1-es bitek feletti számértékeket összeadva készen áll a decimális megfelelő. Az eljárást illusztráljuk az [A] karakter első byte-ján:

128	64	32	16	8	4	2	1
0	0	0	1	1	0	0	0

16 + 8 = 24 , tehát a byte értéke decimálisan 24. A byte értékét POKE utasítással közvetlenül be lehet írni a RAM bármely címére, így létrehozhatunk tetszés szerinti új karaktereket. A gép azonban billentyű-leütés hatására a ROM-ból keresi ki a megfelelő karaktert, ezért utasítanunk kell arra, hogy máshonnan vegye. Két címen kell a memória tartalmat módosítani aszerint, hogy a RAM-ban hol hoztuk létre az új karakterkészletet. A két cím: 65298 és 65299. A gép karakterkészlete azonban 2-szer 256 karakterből áll, ennyit újraírni óriási munka lenne. Ezért az a szokás, hogy a ROM-ból RAM-ba másoljuk át a meglévő karakterkészletet, majd a szükségeseket módosítjuk.

A C16 és plus/4 gépek egy TEDMON nevű beépített gépi nyelvű programmal rendelkeznek, amely igen egyszerű lehetőséget ad arra, hogy a felhasználó gépi kódu programot írjon. A TEDMON egyetlen karakterből álló parancsaival azonban különösebb programozói ismeretek nélkül is lehetővé válik a memóriacímek közvetlen olvasása (ROM is!), írása, elmentése, betöltése. A TEDMON-t BASIC-ből a MONITOR parancs kiadásával lehet elérni. A parancs hatására kiírja a regiszterek tartalmát, majd várja a

parancsokat. A TEDMON-parancsok paramétereit azonban hexadecimális számként kell beírni, ezért ki kell térnünk a 16-os számrendszerre röviden. E számrendszer az alábbi számjegyeket használja:

0 1 2 3 4 5 6 7 8 9 A B C D E F

A betűket mindig [SHIFT] nélkül kell írni, az éppen használatos üzemmódban. Szerencsére a C16 és plus/4 gépek BASIC-je tartalmaz két függvényt (HEX$ és DEC) amelyekkel átírhatunk decimális számot hexadecimálissá és fordítva:

PRINT DEC("3C") parancs kiírja: 60 (=3C decimálisan)
PRINT HEX$(60) parancs kiírja: 3C (=60 hexadecimálisan)

A hexadecimális számok elé szövegben $ jelet szokás tenni, pl: $5000, $D7FF.

A TEDMON egyik parancsával (TRANSFER) lehetséges egy egész tártartomány átírása egy megadott címtől kezdődő új helyre. A parancs formája:

T <1.cím> <2.cím> <3.cím>

Az 1.címtől a 2.címig terjedő tartományt írja át a 3.címtől kezdődően.

A karaktermódosítás sorrendje:

- Karakterkészlet áthelyezése MONITORban
- Módosított karakter byte-jainak beírása (BASIC)
- Mutató beállítás az új helyre (BASIC)

Példaként bemutatjuk az alsó indexbe kerülő 2-es szám kialakítását.

128	64	32	16	8	4	2	1	byte(dec.)	cím(dec.)
0	0	0	0	0	0	0	0	0	10216
0	0	0	0	0	0	0	0	0	10217
0	0	0	0	0	0	0	0	0	10218
0	1	1	0	0	0	0	0	96	10219
1	0	0	1	0	0	0	0	144	10220
0	0	1	0	0	0	0	0	32	10221
0	1	0	0	0	0	0	0	64	10222
0	1	1	1	0	0	0	0	112	10223

A karakterkészletet (53248 - 57343) a 8192($2000)-vel kezdődő címre toljuk el az alábbi TEDMON-paranccsal:

T D000 D7FF 2000

Amennyiben csak a kis/nagy betűs készletre van szükségünk (ebben vannak a gyárilag ékezetesített betűk is), akkor $D000 helyett $D400 az első cím. Olyan kezdőcímet választottunk, hogy az új készlettel írandó BASIC program számára is maradjon elég hely $1000 és $2000 között (a BASIC-programot $1000-től írja be a RAM-ba a gép). A címeket úgy választottuk, hogy a C= gombbal együtt nyomott [x] (grafikus karakter, amire nincs szükségünk) adja az alsóindexbeli kettest. A megfelelő címeket a képernyőkódok (gépkönyv) alapján (és némi próbálkozással...) lehet kiszámítani. A karaktergenerátor ROM ugyanis e kódok sorrendjében tárolja az egyes karaktereket, s természetesen az áthelyezés után is megmarad a sorrend (egy karakter: 8 byte!). Az első karakter:@ ("kukac"-nak nevezik a programozók), erre ritkán van szükség, ezért kezdhető itt a módosítás (könnyebb a címeket kiszámolni).

A karakterkészlet MONITOR-ban történő áthelyezése után az [X] paraméter nélküli paranccsal visszatérve BASIC-be az alábbi kis programmal írhatjuk be a látott egy karaktermódosítást:

```
100 POKE 10216,0
110 POKE 10217,0
120 POKE 10218,0
130 POKE 10219,96
140 POKE 10220,144
150 POKE 10221,32
160 POKE 10222,64
170 POKE 10223,112
180 POKE 65299,33:POKE 65298,192
190 END
```

A program 180. sora végzi a mutatók beállítását az új címre. A 65298-as címre beírt 192 annyit közöl a géppel, hogy ezután ne a ROM-ból, hanem a RAM-ból vegye a karakterek képét. A 65299-s cím ($FF13) tartalma jelöli ki az új karakterkészlet helyét. Érdemes kissé pontosabban megnézni ezt az úgynevezett TED vezérlő regisztert, mert használatának ismeretében bárhová tehetjük a karakterkészletet. A gép bekapcsolásakor e cím tartalma: 209. Bitekre bontva:

128	64	32	16	8	4	2	1	
1	1	0	1	0	0	0	1	-> 209

A regiszter 2.-7. bitje tartalmazza a karaktergenerátor kezdőcímét (felső byte)

1	1	0	1	0	0	0	-> 208

A 208 értéket hexadecimálissá alakítva $D0 adódik, ami a karakter ROM $D000-al kezdődő 4 kilobyte-os területére állítja be a mutatót. $D00 helyett $2000-től kezdve a karakterek képeit, $20-at kell beírni, ami decimálisan: 32. A megfelelő bitek:

0	0	1	0	0	0

Ne felejtsük el, hogy a regiszter 0. és 1. bitje más célt szolgál, változatlanul kell hagyni, tehát az egész byte:

0	0	1	0	0	0	0	1	-> 33

Ezért írtunk a 65299-es címre 33-at.

A C16 és plus/4 gépek grafikus képernyő használata esetén nem $1000, hanem $4000 címtől tárolják a BASIC-programot, az $1000 és $4000 közötti területet a grafikus képernyő működtetése lefoglalja, ezért az előzőekben átmásolt karakterkészlet is törlődik. Grafikát igénylő program tervezése esetén jobb a karaktereket magasabb, pl. az $5000 címtől kezdve tárolni. Ebben az esetben az előző módszerrel kiszámítva a 65299-es címre 81-et kell írni, annak a grafikus karakternek, melyet alsóindexbeli kettesre módosítottunk 22504-es cím az első byte-ja.

Az így módosított karakterkészletet kimenteni a TEDMON megfelelő parancsával lehet:

S <"név"> <egység száma> <első cím> <második cím>

Kazettára kimentés tehát:

S "karakter" 1 2000 2800

Ez a parancs "karakter" néven szalagra veszi a tártartalmat $2000 címtől $2800 címig. Az így kimentett tártartalmat be lehet olvasni szintén monitorból:

L <"név"> <egység száma> azaz konkrétan:
L "karakter" 1

A betöltés lehetséges a megszokott BASIC-paranccsal is, de ekkor ott is használni kell egységszámot és utána egy 1-est, ami utasítja a gépet, hogy mindent arra a címre írjon, ahonnan ki lett mentve:

LOAD "karakter",1,1

Az új karakterkészletet használó program írása a karakterkészlet áthelyezése, illetve a módosított készlet betöltése után a szokásos módon történhet az alábbiak figyelembe vételével:

- a program elején be kell állítani a TED-regisztert, mert ez nincs benne a kimentett tártartalomban (180. sor az előző listában)
- ha grafikát használ a program (új karakterkészlet $5000-től!) célszerű előtte egy GRAPHIC 1,1:GRAPHIC0 parancsot kiadni, ekkor eleve $4000-től tárolja a programot.

A kész programot a megszokott módon lehet kimenteni, de betöltése előtt mindig be kell tölteni a karakterkészletet is. Amennyiben programunk $4000-nél kezdődik, lehet a karakterkészlettel együtt kimenteni MONITORból:

S "név" 1 4000 5800 (kazettára)
S "név" 8 4000 5800 (lemezre)

Ilyenkor betöltés előtt ki kell adni egy GRAPHIC1,1:GRAPHIC0 parancsot, ebből a gép tudja, hogy $4000-nél kezdődik a program. A betöltés:

L "név" 1 (kazettáról)
L "név" 8 (lemezről)

A BASIC is ad lehetőséget arra, hogy a betöltendő program ugyanazokra a címekre kerüljön, ahonnan ki lett mentve:

LOAD"név" 1,1 (kazettáról)
LOAD"név" 8,1 (lemezről)

Az eddig látott eljárásnak az a szépséghibája, hogy a módosított karakterkészlet előtt nem fér el túl hosszú program, esetleg a program "belelóg" a karakterkészletbe és elrontja. Ezért gyakorlott programozók a karakterkészletet $1000-től, tehát a program előtt helyezik el. Ez a megoldás annyival bonyolultabb, hogy ilyenkor a program-mutatót is át kell állítani, mert különben a gép $1000 címen keresi a programot. Az eljárás műveletei:

- Monitorba kapcsolás
- Karakterkészlet áthelyezés:
  T D000 D7FF 1000
- A leendő program helyére 0 byte-okat tenni:
  F 1800 4000 0
- Visszakapcsolás BASIC-be:
  X
- A karaktermutatók beállítása:

POKE 65299,16:POKE 65299,33
- A programmutató beállítása:
POKE 44,24:POKE46,24

Ezután lehet programot írni, vagy betölteni és bármikor a karakterkészletet módosítani. Az így megírt programot MONITOR-ból kell kimenteni:

S "név" 8 1000 3000 (lemezre)
S "név" 1 1000 3000 (kazettára)

A fenti adatok az $1000-$3000 közti tártartalmat mentik ki, de lehet ennél hosszabb is a program, akkor nagyobb területet kell kimenteni. Monitorban meg lehet állapítani a megírt program által elfoglalt címtartományt az [M] parancs segítségével:

M cím : a címet követő 96 byte tartalmát kiírja a képernyőre. Addig kell folyamatosan keresni, amíg egymás után három nullát találunk, itt a program vége.

Az így készített program betöltéséhez érdemes egy betöltő programot írni, amely átállítja a mutatókat, majd betölti a karakterkészletet is tartalmazó programot:

```
10 KEY1,"POKE65298,33:POKE65299,16"+CHR$(13)
20 POKE44,24:POKE46,24
30 POKE65298,33:POKE65299,16
40 LOAD"név",1,1
```

A program 10-es sora az F1 funkcióbillentyű tartalmává teszi a karaktermutatók beállítását és egy [RETURN] billentyűnyomást. Ezáltal, ha elszáll a kép - ami előfordulhat - gombnyomásra visszajön.Ezt természetesen a programnak is közölnie kell a felhasználóval. A fenti programot betöltve és elindítva a mutatók átállítása után betölti a "név" programot kazettáról és el is indítja. A program elindításakor furcsa zavaros képet látunk, mert a karaktereket üres helyről keresi a gép, majd a program töltésekor gyorsan, de látható fokozatossággal megjelennek a betűk - ahogy a programból helyükre kerülnek a karakterek képei. Természetesen ki is lehet kapcsolni a képernyőt erre az időre: POKE65286,239, visszakapcsolás: POKE65286,27.

Gyakorlott programozói fogásokkal mindezt lehet másképpen is végezni, de könyvünk célja, hogy a legkevesebb (BASIC alapjai) programozói ismerettel is képessé tegyen jó oktatóprogramok írására - ezért nem mehetünk bele mélyebben ebbe a témába.

A fenti eljárás nemcsak a karakterek módosítására használható, hanem egy gyárilag ékezetesített gép karakterkészletének egy nem ékezetesített gépre való átvitelére, azaz gépek házi ékezetesítésére. Az ékezetesítő programot (melyeket készen is lehet kapni szaküzletben) minden bekapcsolás után be kell tölteni.

Külön meg kell említenünk a C64 tulajdonosok körében igen népszerű SIMON'S BASIC nevű programot, amely sok új BASIC utasítás használatát teszi lehetővé, nagyszerűen lehet vele rajzolni, és tetszés szerinti karaktereket kialakítani. Szaküzletekben kevéssé kapható, de a legtöbb C64 felhasználó szívesen tovább adja új érdeklődőknek.

A TVC gépek összesen 192 karaktere közül 64 bekapcsoláskor szóköz, e karaktereket a felhasználó szerkesztheti meg programból. Egy karakter létrehozása egyetlen BASIC utasítással egyetlen programsorral lehetséges! E felhasználó által definiálható karakterek karakter-kódja 160-223 között lehet.

A TVC gépeken egy karakter 10X8-as pontmátrix, tehát betűi magasabbak a Commodore karaktereknél, és egy karakter képe így 10 byte-ot igényel.Az előzőekben látott alsó indexbeli kettes képe:

128	64	32	16	8	4	2	1	byte (dec.)
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	96
1	0	0	1	0	0	0	0	144
0	0	1	0	0	0	0	0	32
0	1	0	0	0	0	0	0	64
0	1	1	1	0	0	0	0	112

Az új karaktert definiáló utasítás:

```
SET CHARACTER 162,0,0,0,0,0,96,144,32,64,112
```

Ezen utasítás végrehajtása után a 162-es karakterkód az alsó kettest jelenti. Kiíratható két módon:

```
PRINT CHR$(162)
<ALT>2
```

Ez utóbbi didaktikailag kitünő megoldás: az <ALT> gombbal együtt nyomott 2-es szám alsó indexbe írja a 2-est. Az egyes karakterkódok billentyűknek való megfeleltetését a géphez adott táblázat tartalmazza. Újabb számítás nélkül alkothatunk felső indexbeli kettest az első és a második öt byte megcserélésével:

```
SET CHARACTER 138,96,144,32,64,112,0,0,0,0,0
```

Ezáltal a <CTRL> gombbal nyomott kettes felső indexbe kerül.

Egy karakter rajzolt figurák számára elég kicsi, de több módosított karakter egymás mellé írásával nagyobb alakzatot is lehet létrehozni. Mint később látni fogjuk, több karaktert együtt mozgathatunk, ezzel a lehetőségek szélesre nyílnak.

A karakterszerkesztés összefoglalásaként elmondhatjuk, hogy a munka - TVC kivételével - fáradságos, de megéri. A saját igény szerint tervezett karakterek új lehetőségeket nyitnak meg a programíró számára:

- indexszámok, speciális karakterek (görög, cirill betűk, gyökjel, stb.) használata
- játékokhoz tetszőleges figurák, melyek mozgathatók is
- szimulációs programokhoz atomok, ionok, sejtek, stb.
- kottán elhelyezhető, mozgatható hangjegyek
- drámarendezéshez színészek mozgatható figurái
- sakkfigurák

7.1.3. Táblázatok

Több tantárgyban az információs modul gyakran igényli adatok, tények, tulajdonságok táblázatos összefoglalását, illetve megfigyelendő adatok táblázatos közlését a törvények, szabályok "felfedezése" céljából. A Commodore gépek sok grafikus karakterrel rendelkeznek, melyekből változatos táblázatokat és ábrákat lehet felépíteni, míg a PRIMO gépen a táblázatokat grafikusan kell elkészíteni. Három egyszerű módszer javasolható a táblázatok kiíratására:

- PRINT utasításokkal, szóközökkel.
- PRINT utasításban használt TAB(n) függvénnyel írni megfelelő helyre a grafikus karaktereket.
- POKE, vagy CHAR utasításokkal a grafikus karakterek képernyő pozíciókra való elhelyezésével.

Az első módszer előnye, hogy programírás közben látszik a táblázat formája, azonnal lehet módosítani. A másik két módszernél négyzetrácsos papíron kell a rajzot megtervezni, a megfelelő képernyőpozíciót kiszámolni. A PRINT utasítás esetén gondolni kell a már képernyőre vitt és ott maradó szövegre is - mert ez az utasítás folyamatosan írja a sorokat, ha szükséges felfelé "tolja" a képernyő tartalmát -, vagy táblázatkészítés előtt törölni a képet. A POKE utasítás a képernyőn lévő karaktereket felülírja, így arra is alkalmas, hogy egy már ott lévő szövegben bizonyos részek törlésével más részeket bekeretezzünk, vagy egyéb módon kiemeljünk.

Didaktikai szempontból nem előnyös a számítógépek számkiírási formátuma: a tizedesvessző helyett pontot kell használni és változó, vagy számolt adat kiírásánál a tizedespont előtt nem ír ki nullát egynél kisebb számoknál: .567 = 0.567 (PRINT utasítással természetesen kiírathatjuk- PRINT"0.567"). Egész számok esetén is szükség lehet az ilyen kiíratásra : 45.00 , ha pl. a pontosságot érzékeltetjük vele. A tizedespont használatát meg kell szokni (sok országban használják így kézírásban is), a másik problémán szerencsére egyszerű BASIC utasításokkal lehet segíteni: a PRINT utasításban alkalmazott STR$ és MID$ függvényekkel (Commodore és PRIMO).

Az STR$(A) függvény értéke az (A) aritmetikai kifejezés (pl. egy változó) értékének sztring alakját adja meg. A MID$ sztring-függvény az első paramétereként megadott sztring bizonyos karaktereiből új sztringet állít elő. például:

```
A$="molekula" , akkor
MID$(A$,2,4) = "olek"
```

Látható, hogy a második paraméter az első paraméterként megadott sztring azon karakterhelyét jelöli, ahol kezdődik az új sztring, a harmadik paraméter pedig az új sztring hosszát. A harmadik paraméter elhagyható, ekkor az eredeti sztring végéig tart az új (maximum 255 karakter). Egy rövid programlistán bemutatjuk e függvények alkalmazását:

```
10 A=23
20 PRINT STR$(A)+".00"
30 A=0.632: PRINT A
40 B$="0"+MID$(STR$(A),2): PRINT B$
```

A 20. sor így írja ki A értékét: 23.00
A 30. sor bemutatja a 0 nélküli kiírást: .632
A 40. sor úgy hozza létre B$ sztringet, hogy egy nullát ír a tizedespont elé, a MID$ "levágja" az egyébként tizedespont előtt mindig ott levő szóközt. Az eredmény: 0.632 .

Gyakran szükség van egy számadat kerekítésére. Legegyszerűbb módja pl. két tizedesre kerekítésnél: a számot százzal megszorozzuk, hozzáadunk 0.5-et, egész részét képezzük és elosztjuk 100-al. Próbáljuk ki:

```
10 INPUT X
20 X=INT(X*100+0.5)
30 X=X/100
40 PRINT X
```

Amelyik gépek ismerik a formázott kiíratást biztosító PRINT USING utasítást (C16, Plus/4, PRIMO, TVC), ott nem szükséges a kerekítés fenti megoldása.

```
PRINT USING"$##.##";A
```

Ez a programsor (A) változó értékét két egész számjegy utáni két tizedesre kerekített formában írja ki. Ha pl. A=3.2782, akkor a kiírás:

```
*3.28
```

Látható, hogy a formátumot szigorúan betartva, a hiányzó számjegy helyett csillagot ír. Egynél kisebb szám esetén a tizedespont elé 0-t ír. Amennyiben kevés helyet foglalunk #-jelekkel, akkor hiba történik, ezért gondosan kell figyelni a helyfoglalás és a változó értéke közti megfelelésre, inkább több helyet foglaljunk.

Az előző fejezetben említettük a képváltások gyakoriságát, módját. Leghelyesebb, ha a tanuló gombnyomással kérheti a továbbhaladást amikor végzett a képernyőn levő anyag feldolgozásával. Ennek a megoldásnak a technikai kivitelezése igen egyszerű, de gépenként más. A Commodore gépeken használható a GET utasítás az alábbi módon:

```
20 GET A$: IF A$=" "THEN 20
```

Mindaddig ezt a sort "ismétli" magában a gép, amíg bármely billentyűt le nem nyomjuk, utána a következő programsorra lép, ahol képernyőtörlés után írathatjuk az újabb szöveget. A C16 és plus/4 gépek ismerik a GETKEY utasítást, amely az előző sort helyettesítheti:

```
20 GETKEY A$
```

Addig fog állni a program, amíg le nem nyomunk egy billentyűt.

A PRIMO és TVC gépeken az INKEY$ utasítás alkalmas erre a célra, amely hasonló a GEThez :

```
20 A$=INKEY$: IF A$="" THEN 20 ELSE 30
```

Az INKEY$ utasítás mintát vesz a billentyűzetről, és ennek eredménye lesz A$ értéke. A mintavevés addig folyik, míg egy billentyűt le nem ütünk. A TVC gépeken használható a GET változó nélkül is:

```
20 GET
```

Addig vár a program, amíg le nem nyomunk egy billentyűt.

Nem szabad elfelejteni az első egy-két alkalommal érthetően közölni a továbbhaladás módját a felhasználóval, amíg meg nem szokja. Például: " Ha a jobb alsó sarokban a TOVÁBB? feliratot látod, akkor bármelyik gombbal továbbhaladhatsz."

A mikroszámítógép lehetőséget ad tetszőleges, szép táblázatok, grafikonok bemutatására. Érdemes e lehetőséget alaposan megismerni, mert több didaktikai feladat megoldását eredményesen segítheti (motiváció, tényanyagnyújtás, általánosítás, alkalmazás, gyakorlás, stb.).

7.2. A kérdező modul (kérdés–válasz)

E modul része a kérdésfeltevés, a válaszhoz szükséges útmutatás és a válasz fogadása – feleletválasztás esetén a válaszlehetőségek felsorolása.

A kérdések megfogalmazásában körültekintőbben kell eljárni a tanitási órákon megszokottnál, mert itt nem lehet közben módositani a kérdést sikertelen válaszok miatt, legfeljebb a rossz válasz utáni elágaztatás vezeti új kérdéshez a tanulót. A feleletválasztásos módszer alkalmazásánál az irásbeli tesztekhez hasonlóan fogalmazhatunk. Feleletalkotást igénylő kérdéseknél fontos a rövidség, tömörség, egyértelműség. A tanulónak itt teljesen önállóan és precizen kell válaszolnia, nem úgy, mint szóban, vagy irásbeli dolgozatban, ahol egy kis pontatlanság még nem biztos, hogy hiba.

A feleletválasztásos kérdéseknél a megszokott módon felsorolhatjuk a válaszlehetőségeket számokkal, vagy betűkkel jelölve: 1 2 3 4, A B C D. A tanulói válasz fogadására két utasitást használhatunk: INPUT és GET (GETKEY). Az INPUT előnye, hogy a leütött betű mindaddig javitható, amig [RETURN] billentyűvel nem jelzi a válasz eldöntését a tanuló. A GET használata annyival egyszerűbb, hogy csak a billentyűt kell leütni [RETURN] nélkül – ez egyúttal a hátránya is: nem javitható a téves leütés. Az INPUT a legtöbb gépen azonos módon használható:

```
10 INPUT "Válaszod:"; A$
```

TVC gépnél, ha szöveget is kivánunk közölni az INPUT után, akkor az INPUT PROMPT utasitást használhatjuk:

```
10 INPUT PROMPT"Válaszod:":A$
```

A program a [RETURN] leütéséig vár, és minden leütött karakter A$ sztring része lesz (betű, szám és szóköz is). A vezérlő billentyűk (pl.kurzormozgatók) közben működtethetők – erre vigyázni kell, de ki is használhatjuk (ld. később).

A GET illetve GETKEY utasitás eltérő az egyes gépeken. Mint az előző fejezetben már egy alkalmazásnál láttuk, a GET a billentyűzetről mintát véve a mintavétel értékét adja eredményül, akkor is ha nem ütünk le semmilyen billentyűt (üres karakter). Ezért gondoskodni kell róla, hogy, hogy a program ne haladjon tovább, amig valamilyen karakter nem lesz a minta értéke. Ennek szokásos megoldása:

```
20 GET A$: IF A$= "" THEN 20
```

A PRIMO és TVC gépek a GET helyett az INKEY$ utasitást használják:

```
30 A$=INKEY$: IF A$="" THEN 30
```

A C16 és Plus/4 gépen:

```
40 GETKEY A$
```

A fenti megoldások teljesen megegyeznek az információs modul képváltásaira javasolttal, ezért vigyázni kell a változók elnevezésére: INPUT utasitásnál a karakter nélkül leütött [RETURN] esetén a kért változó előző értéke megmarad – lapváltásnál bármelyik gomb lenyomását megengedtük! Legjobb képváltásra mindig ugyanazt a – másra nem használt – változót igénybe venni (pl. T$).

Lehetséges úgy kivitelezni a választást, hogy a program kiírja a választható betűket, számokat és a kurzormozgató gombokkal kell a kívánt helyre vinni a kurzort, majd [RETURN]-el jelezni a döntést. Az alábbi kis program ezt a megoldást mutatja be:

```
50 PRINT"A<crsrle><crsrbalra>B<crsrle><crsrbalra>C"
60 INPUT A$
70 PRINT "<CLR>"
80 PRINT"Válaszod:"; A$
```

Futtatáskor egymás alá irja a három választható betűt (A B C), majd a választás ([RETURN]) után töröl és kiírja a döntés eredményét.

Minden esetben gondoskodni kell a szabálytalan válaszra történő reagálásról. Ha a választást tartalmazó változó értéke nem a megengedettek valamelyikével megegyező, akkor erre figyelmeztetni kell a tanulót és visszairányítani újraválaszolásra. Ennek érdekében a válaszértékelő modul részét képező (ld.ott) IF...THEN utasítást tartalmazó sorok után kell írnunk:

```
100 PRINT"Kérlek a megadott betűkkel válaszolj!": GOTO 60
```

Így, ha a tanuló válaszra használt karaktere nincs az IF...THEN sorokban, akkor a 100. sorra kerül a program és a figyelmeztető mondat után újra várja a választ (60-as sor). Ha a képernyő megtelt, akkor szükség van képváltásra, ezért a 100-as sorba törlő utasítást kell írni, és a kérdés kiíratására küldeni vissza:

```
100 PRINT"<CLR>"
110 PRINT"Kérlek, a megadott betűkkel válaszolj!"
120 GOTO 10
```

Az önállóan szerkesztett válaszok fogadása szintén az INPUT és GET utasításokkal oldható meg. E két utasítás alapvető alkalmazását már említettük, itt kitérünk néhány hasznos módosítási lehetőségre, előbb azonban meg kell ismerkedni a billentyűzet gép általi olvasásával.

A Commodore gépek egy tíz karakter hosszúságú billentyűzet pufferrel rendelkeznek, amely a leütött, de még fel nem használt karakterek tárolására szolgál. E pufferbe a program futása közben is lehet karaktereket bevinni, melyek például egy soron következő INPUT vagy GET utasításnál használódnak fel. Egy oktatási programnál ez komoly gondokat okozhat: a szöveg olvasása közben véletlenül megnyomott billentyűk a pufferbe kerülve egy INPUT vagy GET útján változók nem szándékolt értékeivé válnak - így például folyamatosan továbblapoznak a "Tovább?" jelzésnél, stb. Alaphelyzetben a billentyűk ismétlik önmagukat folyamatos nyomvatartás esetén: egy elgondolkozó tanuló ujját a billentyűn felejtve zavart okoz a programban. Szerencsére a billentyűzet puffer beállításai megváltoztathatók POKE utasításokkal. A memóriahelyek (decimálisan):

C16, Plus/4	C64	
239	198	a pufferben lévő karakterek száma
1343	649	a puffer mérete (0 - 10)
1344	650	az ismétlés módja
1319-1328	631-640	a billentyűzet puffer

A billentyűzet ismétlés módjánál az alábbi értékek fontosak:

0 csak a kurzormozgató, léptető billentyűk ismételhetők
64 semmilyen billentyű nem ismételhető
128 minden ismétlés újraengedélyezése

Legcélszerűbb a program elején a betűk ismétlését letiltani, de a léptetéseket nem :

```
10 POKE 1344,0    ( C16, Plus/4)
10 POKE 650,0     ( C64)
```

Vigyázat: a billentyűzet puffer hosszát 0-ra állítva gyakorlatilag letiltjuk a billentyűzetet, utána már csak a RESET, vagy a kikapcsolás segít, hiszen semmilyen billentyű sem kerülhet a pufferbe!

A billentyűzet puffer címeire POKE utasítással bármilyen karaktert elhelyezhetünk programból. Így például segítségképpen egy kérdésnél a válasz kezdőbetűjét beírhatjuk a tanuló helyett.

Az INPUT utasítás néhány fontos tulajdonsága:

- A legtöbb karaktert a változó részének tekinti, kivétel: idézőjel, kettőspont, vessző.
- A vesszőt és kettőspontot két változó értéke közötti elválasztó jelnek tekinti. Idézőjelek között azonban ezeket is karakternek fogadja el.
- A kérdőjeltől a [RETURN]-ig terjedő karaktereket (max.88) dolgozza fel.
- Ha nem a változónak megfelelő értéket kap, pl. numerikus változót kérdezve betűt írunk - akkor ?REDO FROM START üzenettel újra várja a megfelelő értékeket.

Többször jó lenne, ha a tanuló válaszában használhatna kettőspontot, vagy vesszőt anélkül, hogy idézőjelet kellene írnia. Ezt elérhetjük úgy, hogy a billentyűzet puffer hosszát 1-re állítva beviszünk programból egy idézőjelet: az INPUT a kérdőjel után az idézőjelet kiírja, s az utána következőket sztringnek tekintve elfogadja a vesszőt és a kettőspontot is. Az alábbi kis program az előbbieket mutatja be (C16, Plus/4):

```
10 POKE 239,1
20 POKE 1319,34
30 INPUT A$
40 PRINT A$
```

Lehetőség van a kérdőjel kiírásának letiltására is, ha zavaró :

```
10 POKE 19,1
```

A GET utasítás csak egy karakter beolvasására használható. Mivel a sztringeket össze lehet adni, így sorozatban alkalmazott GET utasításokkal szavakat is lehet bekérni:

```
10 REM " C16 és Plus/4-re"
20 PRINT"CLR":PRINT:PRINT
30 GETKEY A$:PRINT"<crsrfel>"A$
40 GETKEY B$:PRINT"<crsrfel><crsrjobbra>"B$
50 GETKEY C$:PRINT"<crsrfel><crsrjobbra><crsrjobbra>"C$
60 X$=A$+B$+C$
100 REM" C64-re"
110 PRINT"CLR":PRINT:PRINT
120 GET A$:IF A$="" THEN 120
125 PRINT"<crsrfel>A$
```

```
130 GET B$:IF B$="" THEN 130
135 PRINT"<crsrfel><crsrjobbra>B$
140 GET C$:IF C$="" THEN 140
145 PRINT"<crsrfel><crsrjobbra><crsrjobbra>C$
150 X$=A$+B$+C$
```

Az egy-egy GET-utasítással bekért karaktereket a kurzormozgató vezérlőkarakterek segítségével úgy írathatjuk ki, mintha folyamatosan írta volna a tanuló válaszát egy INPUT-ra. X$ több egykarakteres sztring összege is lehet. E megoldás lehetővé teszi, hogy a választ betűnként külön-külön értékeljük, mert A$, B$, C$ értéke az összeadás után is megmarad. Kétségtelenül elég bonyolult és hosszadalmas ez a GET-felhasználás, de bizonyos pl. nyelvi feladatokban hallatlan előnyt jelent a betűnkénti válaszértekelés lehetősége. A sztringek összeadása előtt is lehet értékelni a beírt egy karaktert: így kiszűrhetők a meg nem engedett karakterek.

A válasz fogadása akkor okoz újabb problémákat, ha alsó vagy felső indexet kell használni a válaszban a tanulónak (fizika, kémia, matematika). Az előző fejezetben látott karaktermódosítás jelenti a "legelegánsabb" megoldást az információs modulban. A válaszadásnál is jó ez a módszer, de problémákat vet fel. Az átdefiniált karakterek jelentését nehéz és nem célszerű egy-egy program erejéig billentyűre ragasztott cédulával jelölni. A tanulónak gondolkodás és válaszolás közben fokozottan kell figyelni a billentyűhasználatra. Többnyire az adott program, illetve a felhasználók ismerete döntheti el a legjobb megoldást, ezért itt még két lehetőséget ismertetünk.

Commodore gépeken INPUT utasításra történő válaszadásnál a kurzormozgató billentyűk eredeti funkciójuknak megfelelően használhatók, vagyis a tanuló egy képlet beírásakor természetes módon léptethet le, vagy fel az indexszám írásához. Az INPUT azonban az egy képernyősorban lévő karaktereket sorrendben alakítja a változó értékévé: egy sorban elférő válasznál nem vesz tudomást a másik sorban lévő karakterekről. A probléma áthidalható két INPUT egymásutáni alkalmazásával:

```
100 print"Ha alsó indexbe akarsz írni számot,"
110 print"akkor a kurzormozgató billentyűvel"
120 print"mozoghatsz lefelé, de az indexszám"
130 print"leírása után léptess vissza!"
140 print" Ha a [RETURN] lenyomása után nem"
150 print"kapsz választ, nyomd meg újra!"
160 print"Az oxigén kémiai jele:"
170 poke 19,1
180 input O$
190 if O$="O" then 210
200 goto 240
210 input "<crsrle>;O
220 if O=2 then 400
230 goto 300
240 if O$="Ox" then 280


400 print"Jól válaszoltál."
```

Itt a válasz fogadása két részre bomlik.Amennyiben az első INPUT-ra jó a válasz (oxigén vegyjele:O), a 190-es sor tovább küldi a programot az indexszámot beolvasó 210-es sorra - egyéb válasz esetén a 200-as sor a vegyjeleket értékelő (240-es sorral kezdődő) részre küldi a programot. A 210-es sor beolvassa az egy sorral lejjebb - tehát

alsó indexbe - írt számot és az O nevű numerikus változó értékévé teszi. A 220-as sor a jó válasz (O=2) esetén a 400-as sorra küld, egyébként a 300-asra, ahol a jó vegyjelű, de rossz indexszámú válaszok értékelése következik. A 170-es sorban a kérdőjel kiírását tiltottuk le, mert a második INPUT azt is beolvasná.

A C16 és Plus/4 gépek rendelkeznek szabadon programozható F1, F2.. billentyűkkel, amelyeket felhasználhatunk a fenti két INPUT összeolvasztására. Próbáljuk ki az alábbi programot:

```
10 poke 19,1
20 key1,CHR$(13)+CHR$(13)
30 input"<crsrle>";A$
40 input"<crsrle><crsrbalra><crsrbalra>";B$
50 print:print"Válaszod:"
60 print A$
70 print" ";B$
```

A 20. sor az F1 billentyűnek kétszeri [RETURN]-lenyomás funkcióját adja, ezért közölni kell a kérdés feladásánál, hogy a képlet leirása után [RETURN] helyett F1-et kell lenyomni. A 60. és 70. sor az eredeti formában irja ki a tanuló által beirtakat. Ez a megoldás olyan képletekre jó, ahol egybetűs vegyjel után indexszám következik:

$$H_2O \quad , \quad H_2SO_4 \quad , \quad H_2O_2 \quad , \quad H_2S \quad , \text{ stb.}$$

Az INPUT és GET utasitásokban sok változót használhatunk egy programban. A BASIC nyelvben háromféle változó ismert:

- numerikus valós változó, értéke pozitiv, vagy negativ szám lehet 9 számjegy pontossággal,
- numerikus egész változó, értéke -32768 és 32767 közé eső egész szám lehet,
- karakterváltozó (sztring), értéke bármilyen, a gép által ismert karakter, vagy karaktersorozat lehet.

A változóknak adott nevet azonositónak szokás nevezni. Az azonositó több karakterből is állhat, de a gép csak az első kettőt veszi figyelembe, ezért nem célszerű több karakteres azonositókat használni. Az azonositó lehet egy betű, két betű, egy betű és egy szám. Az azonositóban jelölni kell a változótipust is:

numerikus valós változók azonositói: A , C2 , DB
numerikus egész változók azonositói: A%, B1% , DD%
karakterazonositók (sztringváltozók): A$, H5$, BB$

A numerikus változók között az egész változó a memóriabeli helytakarékosság célját szolgálja, oktatóprogramban nem szükséges alkalmazni.

A változók elnevezési lehetőségei, mint láttuk igen tágak, hosszú program számára is elegendő változót lehet adni. Minden programbeli kérdésnél azonban nem szükséges új változót bevezetni a válasz fogadására, csak akkor, ha a választ meg akarjuk őrizni az értékeléshez. Már láttuk, hogy az INPUTnál karakter nélkül leütött [RETURN] a változó előző értékét tartja meg, ezért az INPUT utasitás előtt egy értékadó utasitással nullázni kell a változót:

```
100 A=0:A$=""
120 input A,A$
```

Egy hosszú és sokfelé elágazó program készítője könnyen elkeveredhet a sok változó között, ezért legjobb a változókat úgy elnevezni, hogy az azonosítónak legyen köze a változó várt értékéhez.Pl:

```
100 input"Magyarország fővárosa:";BP$
110 input"Franciaország fővárosa:";P$
120 input"A mohácsi csata éve:";M%
130 input"A higany sűrűsége g/cm3-ben:";HG
```

A BASIC nyelv lehetővé teszi mindhárom változótípusnál indexes változók tömbbé szervezését. Az A(0), A(1)....A(n) változók egy tömbbe rendezhetők, ha a tömbnek előre helyet foglalunk a tárban (10-nél kisebb indexig nem kötelező a helyfoglalás):

```
10 DIM A(100), B$(20)
```

Ezzel az utasítással helyet biztosítottunk az A(0)...A(100) és a B$(0)...B$(20) indexes változók számára, létrehoztunk két egydimenziós tömböt. A BASIC megengedi a kétindexes változókat és kétdimenziós tömböket is, így gyakorlatilag kimerithetetlen a felhasználható változók száma. Az indexes változók oktatóprogramokban ritkábban szükségesek, nagymennyiségű adattal való számitásoknál viszont nélkülözhetetlenek. Egy oktatóprogrambeli példát látunk azonban a későbbiekben.

Minden számitógép rendelkezik belső órával, ami bekapcsolástól kezdve működik, két fenntartott változó értékévé téve pillanatnyi állását (Commodore gépek):

- TI valós változó, amely csak olvasható, másodpercenként 60-al nő az értéke, azaz TI/60 a bekapcsolás óta eltelt másodpercek számát adja.
- TI$ egy hat karakteres sztring, két-két karaktere a digitális kijelzésű órákhoz hasonlóan az órát, percet, másodpercet adja meg. Pillanatnyi értékét programból beállithatjuk, igy könnyen használható intervallumok (válaszadási idő) mérésére.

Az alábbi kis programrész bemutatja, hogyan lehet a válaszadás közbeni időt praktikusan kijelezni TI$ segitségével:

```
100 ti$="000000"
110 get b$:print tab(30) mid$(ti$,3,2)":"; mid$(ti$,5,2)
115 if b$="" then 110
120 print"Válaszod:";b$
130 print"Felhasznált idő:";mid$(ti$,3,2)" perc "; mid$(ti$,5,2)" másodperc"
```

A TI$ "nullázásával" elvesztettük addigi értékét, ezért jó előbb elmenteni a későbbiek miatt. Az idő másodpercekbeli kijelzésére alkalmas TI/60 kiiratása, de hogy a válaszra használt időt mutassa, ezért a kijelzett értékből ki kell vonni a kérdés feladásakori értéket. E két módszer megvalósitása:

```
90 x$=ti$:rem"óraállás elmentése"
100 ti$="000000"
110 get b$:print tab(30)mid$(ti$,3,2)":"; mid$(ti$,5,2)
115 if b$="" then 110
120 print"Válaszod:";b$
130 print"Felhasznált idő:";mid$(ti$,3,2)" perc ""; mid$(ti$,5,2)" másodperc"
140 print x$
200 t=ti/60
210 getb$:print tab(30) int((ti/60)-t)
```

```
220 if b$="" then 210
230 print"Válaszod:";b$
240 print"Elhasznált idő:";int((ti/60)-t)"sec"
```

A másodpercek tört részére nyilván itt nincs szükség, ezért használjuk az INT függvényt, amely argumentumának egész részét adja eredményül.

A nagyszámítógépek működtetésére jellemző a multiprogramozás: több program egyidejű futtatása. Természetesen ez csak trükkel lehetséges, hiszen egyidőben csak egy program futhat, ezért időosztásos módszerrel végzik a multiprogramozást: a gép igen sűrűn váltogatja a futó programokat,s a felhasználó úgy érzi, egyszerre futnak. Többnyire azért van szükség ilyen módszerre, mert a nagygép sebességét nem tudják követni a ki- és bemeneti egységek. Bizonyos mértékig egy mikroszámítógép is képes erre a módszerre, így lehet pl. programfutás közben állandóan kijelezni az időt. E program megírása nem könnyű, C64-re készen megtalálható az alábbi ismert könyv 181. oldalán:
Angerhausen, Eglisch, Gerits: Tippek és trükkök a Commodore 64-eshez (DATA BECKER-NOVOTRADE).

Nagymennyiségű adattal dolgozó programok használják a DATA-READ utasításpárt, de oktatóprogramban is van lehetőség érdekes felhasználásukra. A belső (programon belüli) adatbevitel szervezhető meg az utasításpárral:

```
100 data ablak,ajtó,szék,asztal
110 read a$,b$,c$,d$
```

E két programsor eredménye: a$=ablak, b$=ajtó, c$=szék,d$=asztal - azaz a "DATA-sorban" felsorolt adatlistát rendeli hozzá a "READ-sorban" felsorolt változókhoz. A DATA utasításokat a programban bárhol elhelyezhetjük, a READ utasítások végrehajtási sorrendje szerint olvassa ki a program az adatokat. Többnyire a program végére szokták írni a DATA-sorokat. Két dologra kell nagyon figyelni:

- az adatok és a változók száma pontosan egyezzen meg,
- az egymáshoz tartozó adatok és változók típusa azonos legyen (sztring, numerikus).

A DATA-READ utasításpárral olyan kérdéseket szerkeszthetünk, melyekben a tanuló gombnyomással addig kér újabb és újabb adatokat, amíg rá nem jön a megoldásra:

```
100 print"A következő város:"
110 get a$: if a$=""then 110
120 if a$="s" then 160
130 read b$:print b$
140 if b$="Hódmezővásárhely" then 170
150 p=p+1: goto 110
160 input"A megoldás:";c$
170 print"A következő város:"
180 a$="":b$="":c$=""
190 get a$: if a$="" then 190
200 if a$="s" then 240
210 read b$:print b$
220 if b$="Miskolc" then 250
230 p=p+1: goto 190
240 input"a megoldás:";c$
250 print"A következő város:"
```

```
800 data alföldi város,képzőművészet,majolika,
porcelán,mérlegek,Hódmezővásárhely,északi város
810 data hegyvidéki,kohászat,Avas-hegy,Miskolc
```

E programrészlet csak a működést mutatja be, de a kérdések és DATA-sorok cseréjével bármilyen tantárgyi program felépíthető belőle. Elindítás után egy gombot megnyomva újabb adatot ír ki a keresett városról. Ha a tanuló közben rájön a megoldásra, az S betűvel jelezheti, ekkor a program kéri tőle a város nevét (ennek az értékelésével itt nem foglalkozunk). Amennyiben az adatok elfogynak, a program kiírja a keresett nevet, ezzel a tanuló elvesztette ezt a játszmát. A megoldáshoz bekért adatok számát a P változóban gyüjtjük. A 800-810 sorokban vannak a szükséges adatok. Minden adatot vesszővel kell elválasztani, egy adat több szó is lehet: a gép vesszőtől vesszőig olvassa. Akárhány sorba irhatjuk az adatokat, ha a sor elején ott a DATA, akkor a gép folyamatosan olvassa. Ilyen programok alkalmasak versenyre, óra alatti jutalomjátékra, gyakorlásra, és természetesen lehetnek egy nagyobb program részei is.

Foglaljuk össze a kérdés-válasz modul legfontosabb kritériumait, illetve problémáit:

- Kérdés egyértelmű, világos fogalmazása
- A válasz megkivánt formai jegyeinek pontos közlése
- Felkészítés a nem értelmezhető válaszokra
- Billentyűzet puffer beállitása
- Indexszámok beolvasása
- Azonosítók korrekt használata
- Időkijelzés

7.3. A válaszelemző modul

A tanuló válaszadása nem más, mint egy, vagy több változó értékadása. Így a válaszelemző modul feladata lényegében a változó(k) értékének elemzése, az eredmény regisztrálása, visszajelzés a tanuló számára, és az elemzés eredményének megfelelő következmény (továbbirányítás) végrehajtása.

Oktatóprogramban leggyakrabban sztringváltozót szokás alkalmazni a tanulói válaszokra, hiszen a válaszok legnagyobb része szöveges. Számadat fogadására is több esetben előnyösebb a numerikus változónál, pl.:

```
100 print"Egy méter hány centiméter?"
110 input a$
120 a= val(a$)
```

A 100-as sorban feltett kérdésre a tanuló válasza lehet : 100 , de lehet 100 cm is. Ha az INPUT-ban numerikus változót használunk, akkor utóbbi válasz esetén "REDO FROM START" kiírással addig kéri a gép újra a választ, amig szintaktikailag helyeset - azaz csak számjegyekből állót - nem kap. A VAL függvény az argumentumaként szereplő sztring kifejezés numerikus értékét számítja ki, az első számként nem értelmezhető karakterig dolgozva fel a sztringet. Igy pl.:

```
VAL("100 cm") = 100
```

Minden számadat bekérésekor ezt a megoldást használva jelentős hibaforrást lehet kiiktatni.

A számadatok értékelésével kapcsolatban ki kell térni a számítógépek számolási pontosságára, mert kellemetlen meglepetések érhetik ezen a téren a felhasználót. A számítógép nem a matematika, hanem az úgynevezett lebegőpontos aritmetika szabályai szerint korlátozott számú számjeggyel dolgozik (kilenccel). A pontatlanság veszélye jelentős, amikor két igen közeli nagy szám különbségét képezzük, vagy nagy számból nagyságrendekkel kisebb számot vonunk ki : a kilencediken túli értékes jegyek elvesznek. E tulajdonságok miatt sohasem szabad két numerikus változó, illetve numerikus változó és számadat azonosságát feltételül szabni, csak számítás nélküli esetben:

```
100  input"szószám a mondatban:";a$
110  a=val(a$)
120  if a=4 then print"jó!"
130  if a<4 then print"kevés!"
```

Azonosság helyett általában a két számadat különbségének mértékét kell feltételül szabni, legjobb a különbség abszolút értékét használni:

```
100  if abs(a-b)<0.000001 then print"a=b"
```

Az ABS függvény argumentumának (előjeles szám) abszolút értékét adja, tehát:

```
ABS(a-b) = ABS(b-a)
```

Természetesen a program írásakor a várható adatokkal kell kipróbálni a feltételeket a véglegesítés előtt.

A válaszelemzés feladata feleletválasztásos kérdéseknél a legegyszerűbb, mert ekkor csak az általunk megadott (A,B,C,1,2,3..stb.) válaszlehetőségek fogadására kell a programot felkészíteni. Az összes többi esetleges választást egyetlen programsorral el lehet "intézni". Egy ilyen részlet a "Kémiai egyenletek írása" című programból (VORKER):

```
5200 print"<CLR>":print"Hasonlítsd össze papírra írt egyenle-"
5205 print"tedet az alábbiakkal!"
5210 print"Nyomd meg a megfelelő gombot!"
5215 print:print"  C + O -> CO              1"
5220 print:print"  C + O<crsrle>2<crsrfel> -> 2CO          2"
5225 print:print:print"  C + O<crsrle>2<crsrfel> ->  CO<crsrle>2<crsrfel> 3"
5230 print:print:print" 2C + O<crsrle>2<crsrfel> -> 2CO<crsrle>2<crsrfel> 4"
5235 print:print:print" Egyik sem                5"
5250 t$="" :getkey t$
5255 if t$="1"then 5280
5260 if t$="2"then 5300
5265 if t$="3"then 5350
5270 if t$="4"then 5370
5272 if t$="5"then 5320
5275 print"Kérlek, 1, 2, 3, 4, 5-el válaszolj!":goto 5250
```

Az IF...THEN utasításpárok mindig külön sorban kell legyenek, nem úgy mint a legtöbb BASIC-utasítás, amelyekből kettősponttal elválasztva többet is írhatunk egy sorba helytakarékosság céljából. Egy oktatóprogramot azonban gyakran érdemes módosítani a tapasztalatok alapján, ezért az áttekinthetőség miatt jobb csak kevéssé tömöríteni.
A fenti program 5255-5272 sorait egy sorral helyettesíthetjük az ON utasítás felhasználásával:

```
5255 t=val(t$)
5260 on t goto 5280,5300,5350,5370,5320
```

Az ON utasítást követő kifejezés értéke amennyiben 1, akkor a GOTO utáni első sorszámra, ha 2, akkor a második sorszámra lép, stb. Feleletválasztásos kérdések értékelésében mindig ezt a módszert érdemes követni.

Az önállóan alkotott feleletet igénylő kérdések jelentik a számítógép adottságainak igazi kihasználását. Az ilyen kérdéseknek is vannak azonban egyszerű formái, például a nyelvtanulást segítő szótár, illetve szókikérdező programok. Szótárt helyettesítő programban a keresett szót kell beírni, szókikérdező programban pedig a kért szó jelentését. Ezekben a programokban különösen fontos a beírandó szavakra vonatkozó formai utasítás. Természetesen egy ilyen program mikroszámítógépen nem helyettesíthet egy igazi szótárt (nagygépen igen), inkább csak a lehetőség érzékeltetésére szolgál az alábbi lista:

```
5000 rem"szótár"
5010 print"<CLR>:print"Ha befejezted írd be: elég":print
5020 print"A keresett szó magyarul:"
5030 input a$
5040 if a$="szék"then 5100
5050 if a$="anya"then 5110
5060 if a$="apa" then 5120
5070 if a$="elég"then print"Viszontlátásra!":end
5080 print:print"Sajnos ezt a szót nem ismerem!":goto 5020
5100 print:print"németül: der Stuhl":goto 5020
5110 print:print"németül: die Mutter":goto 5020
```

```
5120 print:print"németül: der Vater":goto 5020
5500 rem"szókikérdezés"
5510 print"<CLR>:print
5520 print"Írd magyarul a szavak jelentését"
5530 print"végig kisbetűkkel!":print
5540 print"run"
5550 input"jelentése:";a$
5560 if a$="fuss"then gosub 5800:goto 5600
5570 if a$="fussál"then gosub 5800:goto 5600
5580 if a$="fussatok"then gosub 5800:goto 5600
5590 gosub5900
5600 print"Következő kérdés: boy":a$=""
5610 print:input"jelentése";a$
5620 if a$="fiú"then gosub 5800:goto 5700
5630 gosub 5900
5700 print"Vége a feladatnak.":print
5710 print"Jó válaszaid száma:";jv
5720 print:print"Rossz válaszaid száma:";rv
5730 end
5800 jv=jv+1:print"Jól válaszoltál!":return
5900 rv=rv+1:print"Rosszul válaszoltál!":return
```

A szókikérdező program jó alap lehet a kezdő programiró számára, mert egyszerű kivitelezhetősége, lineáris programfelépitése ellenére jól használható a tanitásban.

Fontos a kérdezett szó jelentésének minden elfogadható szinonimáját szerepeltetni a jó válaszok között. Bármilyen más válasz esetén az 5590-es sorra kerül a program. A válaszelemzést a fentihez hasonló esetben lehet egyszerűsiteni a LEFT függvény segitségével (a * bármilyen karaktert jelenthet):

LEFT (A$,4) = fuss, ha A$="fuss****..."

Általánosan: LEFT (A$,X) az A$ sztring baloldalli X számú karakterét adja eredményül. Igy a program 5560-5580 sorait helyett elég egy sor:

```
5560 if left(a$,4)="fuss" then gosub 5800:goto 5600
```

E megoldással minden olyan szó, vagy mondat is jó válaszként értékelődik, amelyiknek első négy karaktere: fuss. Azért elővigyázatosnak kell lenni: egy tréfás kedvű diák beirhatja: "fuss, ha birsz", s ez is jó válasz lesz!

Ilyen programokban érdemes a jó és a rossz válasz visszajelzését, illetve a jó és rossz pontok számának növelését egy-egy szubrutinra bizni: 5800 és 5900.

A **szubrutin** a programoknak olyan része, amely a program bármely részéből meghivva végrehajtódik, majd a program visszatér a meghivó GOSUB utasitás utánra. Olyan tevékenységet érdemes szubrutinként megirni, amelyiket a program futása közben többször kell teljesen azonos módon használni. A szubrutinok alkalmazásának módjáról a gépkönyvek leirják a legfontosabbakat, itt két dologra hivjuk fel a figyelmet:

- a szubrutin végén mindig ott legyen a RETURN utasitás (nem [RETURN] gomb!),
- a szubrutin végrehajtása után mindig a GOSUB-ot közvetlenül követő utasitásra tér rá a program - ez lehet a GOSUB-al egy sorban is!

Matematikusok és hivatásos programozók gyakran csupa szubrutinból épitik fel programjaikat. Ez a struktúrált programozás felé közelitést jelent - amely nem

elsősorban a BASIC nyelvű programok lehetősége. A strukturált programozás röviden annyit jelent, hogy egy programot teljesen önálló kisebb programokból épitünk fel, amelyek szintén önálló alprogramokból állhatnak.

Az előző programrészletben a jó, illetve rossz válaszok számlálását is a két szubrutin végzi: jv és rv változók értéke nő eggyel-eggyel minden szubrutinhíváskor. A **hibaregisztrálás**nak ezt a megoldását csak akkor használhatjuk, ha minden kérdés, illetve minden hibás válasz után hívásra kerül a szubrutin. Általánosabban az a módszer javasolható, amikor a rossz válaszokat magyarázó szöveges sorban helyezzük el a hibapontok számát növelő értékadó utasitást. Természetesen a hiba súlyossága alapján bármennyivel növelhetjük a változó értékét. Mivel egy mikroszámitógép igen sok változóval tud dolgozni, ezért tetszés szerinti bontásban nyilvántarthatjuk az elkövetett hibákat. A változók elnevezésében itt is célszerű a jelentésre utaló azonositók használata: HP, H1, H2, RV, stb.

Az előbbiekhez hasonló program tartalmazhat a hibás válaszokra vonatkozó különböző magyarázatokat is. Amennyiben e magyarázatokat tartalmazó válaszértékelő modul nem ágaztatja el több irányba a programot, lineáris programnak kell tekintsük, hiszen elágazás csak egy-egy modulon belül van. Az ellenőrző, feleltető programok lehetnek ilyenek de a rossz válaszok magyarázatán keresztül tanitanak is, kisegitő információt adva a kérdés újrafeltevése előtt.

Néhány esetben numerikus változókat használva több feltételezhető rossz választ lehet "csokorba foglalni", igy egyszerűsiteni a programot. Az illető szaktárgy ismerője tapasztalata alapján el tudja dönteni az azonos minősitéssel értékelhető hibák dimenzióit:

```
6000 rem"Évszámok"
6010 print"<CLR>":print
6020 print"Mikor foglaltuk vissza Budát a töröktől"
6030 print"Csak az évszámot ird!":print
6040 input b$
6050 b=val(b$)
6060 if b=1686 then 6200
6070 if abs(b-1686)<2 then 6100
6080 if (b-1686)<4 and (b-1686)>1 then 6120
6090 if (b-1686)<-1 and (b-1686)>-4 then6140
6095 if abs(b-1686) >3 then 6160
6100 print:print"Majdnem jó, csak egy évet tévedtél":end
6120 print:print"Ennél valamivel korábban volt!"
6130 print"Gondolkodj, és válaszolj újra!":goto 6030
6140 print:print"Jó lett volna, de ennél később volt!"
6150 print"Gondolkodás után válaszolj újra!":goto 6030
6160 print:print"Bizony elég sokat tévedtél!"
6170 print"Illene pontosan tudni!":end
6200 print:print"Jól válaszoltál!":end
```

A 6080-as és 6090-es sorban nem használható az abszolút érték, mert a jó dátumnál korábbi, illetve későbbi időpontokat szét kell választani. Még ilyen világos, és a 6030-as sorban tudatositott válasz-irásmód esetén is indokolt az évszám sztringkénti bekérése és a numerikus változóvá alakitás (6050-es sor).

A gyakorló, korrepetáló programok igénylik a legtöbb elágazást, a többszörös lépésszámú mellékutakat. Az alábbiakban egy jellemző programrészlet listáját láthatjuk a "Kémiai egyenletek írása" cimü programból.

```
270 print"<CLR>":print"Emlékezz a magnéziumszalag égésére!":print
280 print"Írd le a kiindulási anyagok nevét!"
290 print"Csupa kisbetűvel irj, és az anyagok neve közé irj kötőszót !"
300 print"Válaszadás után mindig meg kell nyomni"
310 print"a "chr$(18)"RETURN"chr$(146)" gombot!":print
320 print"Kiindulási anyagok nevei :"
330 print:input a$
335 if a$="Mg és O"then 452
336 if a$="O és Mg"then 452
340 if a$="magnézium és oxigén"then 740
342 if a$="magnezium es oxigen"then gosub 9000:goto 320
343 if a$="Magnezium es oxigen"then gosub9000:goto 320
344 if a$="magnezium és Oxigen"then gosub 9000: goto 320
350 if a$="Magnézium és Oxigén"then 740
360 if a$="oxigén és magnézium"then 740
370 if a$="Oxigén és Magnézium"then 740
380 if a$="magnézium és levegő"then 510
390 if a$="magnezium es levegö"then 510
400 if a$="Magnézium és Levegő"then 510
410 if a$="Magnézium és levegő"then 510
420 if a$="levegő és magnézium"then 510
430 if a$="Levegő és magnézium"then 510
440 if a$="levegő és Magnézium"then 510
450 if a$="Levegő és Magnézium"then 510
451 goto 460
452 print"Barátom! Ne siess, még csak az anyagok"
453 print"nevét kérdeztem, nem a vegyjeleket!":a$="": goto 320
460 print: print"Sajnos olyan hibát vétettél, amit nem"
470 print"tudok értékelni. A helyes válasz:":hm=hm+1
480 print" magnézium és oxigén "
490 print:print"                    Tovább?"
500 getkey t$:print"<CLR>":goto 740
510 h1=h1+1:print"<CLR>":print"Rosszul válaszoltál!"
520 print:print"A levegőnek két fő komponense van:     oxigén és nitrogén."
530 print"Melyik lép reakcióba a magnéziummal?"
540 input b$
545 if b$="nitrogen" then gosub 9000:b$="":goto 540
546 if b$="nitrogén" then gosub 9000:b$="":goto 540
547 if b$="oxigen"then gosub 9000:b$="":goto 540
548 if b$="oxigén"then gosub 9000:b$="":goto 540
550 if b$="oxigén"then  print" lgen.":goto 740
560 if b$="Oxigén"then  print" lgen.":goto 740
570 if b$="az oxigén"then print" lgen.":goto 740
580 if b$="Az oxigén"then print" lgen.":goto 740
590 if b$="az Oxigén"then print" lgen.":goto 740
600 if b$="nitrogén"then 670
610 if b$="Nitrogén"then 670
620 if b$="a nitrogén"then 670
630 if b$="A nitrogén"then 670
640 print:  print"Válaszodat sajnos nem tudom értékelni!":hn=hn+1
```

```
650 print: print"                    Tovább?"
660 getkey t$ :print"<CLR>":goto 680
670 print:print"Nagyon rosszul válaszoltál!":print:h2=h2+1
680 print"Már tanultad, hogy az égés oxigénnel"
690 print"való egyesülés."
700 print"Égéskor a magnéziumszalag oxigénnel"
710 print"lép reakcióba."
720 print"                    Tovább?"
730 getkey t$:print"<CLR>"
```

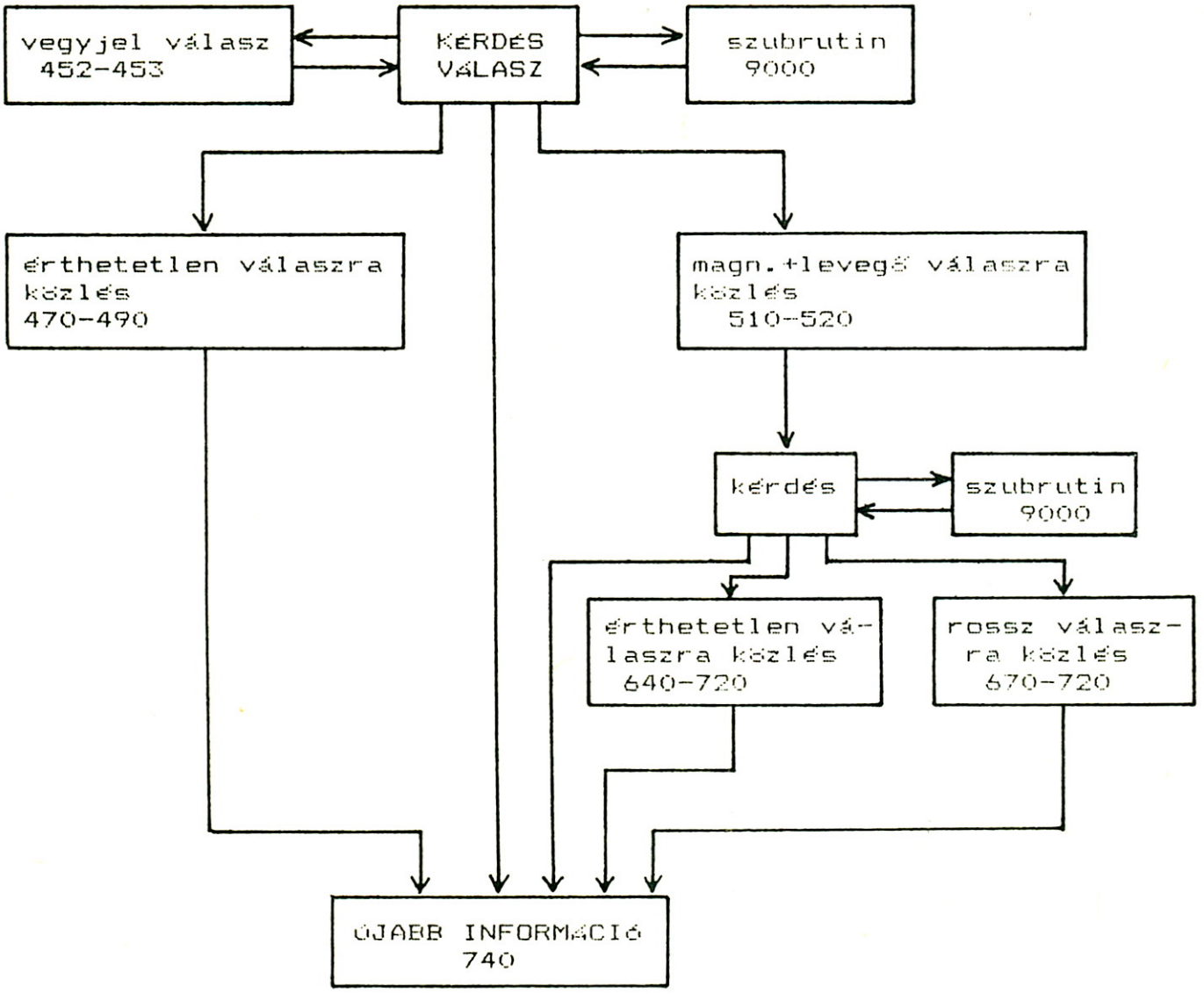
A programrészletben az alábbi modulokat találjuk :

- főág kérdező modul: 270 - 330
- főág válaszelemző modul: 335 - 500
- mellékág információs modul: 510 -520
- mellékág kérdező modul: 530 - 540
- mellékág válaszelemző modul: 545 - 650
- mellékág információs modul: 680 - 730

A "gosub 9000" utasítások egy olyan szubrutint hívnak meg, amelyik újra megmagyarázza az ékezetes betűk használatát. Minden programban érdemes egy ilyen szubrutint használni, és a szöveges választ igénylő kérdések elemzésében gondoskodni az "elfelejtett" ékezetekről (342, 343, 344 és 545-548.sorok). Természetesen ékezethibáért nem célszerű hibapontot adni, bár tanulságképpen egy változóban összegyűjthető az ilyen tévesztések száma. A szubrutint olyan rövidre jó fogalmazni, hogy képváltás nélkül elférjen az újra feladott kérdés "inputjával"

```
9000 he=he+1:print"Sajnos nem jól használtad az ékezetes"
9010 print"betűket! Figyelj jobban, ha szükséges"
9020 print"nézd meg a táblázatot.":return
```

A programrészlet sorszámozása láthatóan nem teljesen egyenletes. Általában 10-el növekedő sorszámokat jó használni, mert így könnyen lehet utólag beszúrni újabb sorokat (a részletben az ékezethibára vonatkozóakat). A C16 és plus/4 gépek RENUMBER parancsával a program befejezése után lehet újrasorszámozni tetszés szerinti növekménnyel (a RENUMBER után semmit sem írva 10-ről indulva 10-esével sorszámoz). Ha programunkat tovább kívánjuk fejleszteni, akkor legalább 10-es növekményt érdemes használni az újrasorszámozásban is. A kihagyott sorszámok természetesen nem terhelik a gép memóriáját, 64000-ig sorszámozhatunk.
Az idézett programrészlet szerkezetét mutatja az alábbi séma:



Az idézett programrészlet szerkezete tulajdonképpen bármilyen tantárgy gyakoroltató, korrepetáló programjának megfelelő. Az új ismeretet közlő, tanitó program annyiban tér el, hogy az információs modulok terjedelmesebbek. Ha jó tanitó programot akarunk irni, akkor az idézetthez hasonlóan - vagy még részletesebben - gondoskodni kell a menetközbeni ellenőrzésről, az ez alapján történő elágaztatásról, kisegitő információkról.

A mikroszámitógép a programiró kezébe adja azt a lehetőséget, hogy több kérdés, feladat eredményét, addigi összpontszámot, kiemelt kritérium-lépés pontszámát is figyelembe vegye a továbbhaladási út kijelölésében:

```
100 if h1>10 and h2>100 and h>200 then 200
110 if h1>20 and h2>100 and h>200 then 210
120 if (h1+h2)>140 and h>200 then 220
```

Az AND ebben az esetben mint egyszerű "és" fogható fel: ha az AND-el összekapcsolt feltételek mindegyike teljesül, akkor kerül a program a THEN utáni sorszámra.

Az IF...THEN sorok lezárásaképpen sosem szabad elfeledkezni az előre feltételezettektől eltérő - így a program számára érthetetlen válaszokról (460-as és 640-es sorok). Mint a programlistán látható az ilyen hibákra fenntarthatunk a hiba helyére utaló kétbetűs azonosítójú változókat, míg az egyéb hibákat egy betű-egy szám kombinációjú azonosítóval jelölve a program futtatása után minden egyes kérdésről megállapíthatjuk az adott válasz milyenségét. A minden kérdés minden válaszlehetőségére vonatkozó "hibanyilvántartás" természetesen csak a programíró számára fontos a változtatások, a program fejlesztése céljából. A rendszeres iskolai használat céljára egy ésszerű mértékig össze kell vonni és egy-egy változóba gyűjteni az egyes hibakat (1, h2, h3, h4, stb.). Az értékelésnél egy-egy változó értéke alapján megfogalmazott részminősítések után a változók összértéke szerint is kell egy záró mondatot mondani:

```
5000 h=h1+h2+h3+h4+h5+h6+he+hn
5010 print"<CLR>:print
5020 print"          É R T É K E L É S":print
5030 if h<1 then 5100
5030 if h<3 then 5150
5040 if h<6 then 5200
5050 if h<10then 5250
5060 goto 5300
5100 print"<RVS ON>G R A T U L Á L O K !":print
5110 print"Tökéletesen dolgoztál, ebben a témakör-"
5120 print"ben mindent tudsz."
5130 print"További sok sikert, viszontlátásra !":end
5150 print"Jól dolgoztál, ezt a témakört lénye-"
5160 print"gében elsajátitottad."
5170 print"További jó munkát, viszontlátásra.":end
5200 print"Sajnos ebben a témában még nem megfe-"
5210 print"lelő a tudásod. Ha van időd, gombnyo-"
5220 print"mással újrakezdheted a programot."
5230 getkey t$: goto 100
5250 print"Bizony gyengén szerepeltél!"
5260 print"Ezt a témát a tankönyvedből újra"
5270 print"kell tanulnod!":end
5300 print"Olyan gyengén szerepeltél, mintha"
5310 print"erről a témáról sosem hallottál volna!"
5320 print"A további teendőket beszéld meg tanároddal":stop
5400 print"h1:"h1;" h2:"h2;" h3:"h3;"h4:"h4
5410 print"h5:"h5;" h6:"h6;" he:"he;" hn:"hn
5420 print"összesen:"h
5430 end
```

A tanuló számára adott minősítés után a program megáll, de CONT paranccsal továbbindítva kiírja a változók értékeit. Itt természetesen az összes kérdés összes válaszlehetőségére vonatkozó változókat kiirathatjuk - melyek értékeit a program fejlesztése érdekében minden tanuló munkája után érdemes begyűjteni. A programban is össze lehetne gyűjteni az egymásután dolgozó tanulók pontértékeit - de a gép kikapcsolásakor a változók értékei elvesznek - így jobb papíron gyűjteni az adatokat.

A tanulói munka értékelésében van legnagyobb jelentősége a stílusnak. A program személyessége az, ami a humanizált gép felé vezet. A finom humor, közvetlenség, a partner személyiségének tisztelete azok a tulajdonságok, melyek egy pedagógus diákok általi megbecsüléséhez vezetnek. Mindezeket kiegészítve a számítógép következetességével, türelmével, sikerre számíthat programunk. A legjobb stílus függ az életkortól,

ezért szöveges program lehetőleg egy korosztálynak szóljon. Szimulációs, demonstrációt segitő programokat azonban lehet úgy készíteni, hogy felső tagozaton és középiskolában is alkalmazható legyen.

A dicséret és a kritika megfelelő adagolása minden program fontos eleme, motivációs szerepe lényeges lehet. A motivációt erősítheti a személyes megszólítás is, amely megoldható egy kis pluszmunkával. A következő programrészlet a kölcsönös bemutatkozás után becézett keresztnevén szólítja a tanulókat.

```
100 print"Szeretettel üdvözöllek Kedves Barátom!"
110 print:print"Én a 132/435. leltári számú Commodore"
120 print"számítógép vagyok, és szeretnélek"
130 print"megismerni. Kérlek, válaszolj néhány"
140 print"kérdésemre."
150 print:input"Vezetékneved:"a$
160 input"Keresztneved:"b$
170 input"Hány éves vagy?"c$
180 if b$="András" then b$="Bandi":goto 1000
190 if b$="Andrea" then b$="Andi": goto 1000
.
.
1000 print"<CLR>":print
1010 print"Kedves "b$"!"
```

Egy programozó talán megbotránkozhat azon, hogy ennyi programsort (tárterületet) ilyen "lényegtelen" célra használunk, de a programirás közben tanulóit szem előtt tartó pedagógus talán nem sajnálja a 100 programsort és a megirására forditott félórát! Amennyiben csak saját osztálya részére készit valaki programot, akkor nem is lesz olyan hosszú ez a rész. A leggyakoribb 50, esetleg 100 nevet feldolgozva a magyar gyerekek döntő többsége megkapja a becenevet. Ha nincs a listában a beirt név, akkor becézés nélkül szólit a b$-ben tárolt néven a program. Többször is érdemes program közben néven szólítani a tanulót, különösen az elmarasztaló és dicsérő értékelésekben. A 150- és 170-es sorokban feltett kérdést figyelem-elterelőnek szántuk, de az életkort fel lehet használni pl. az elmarasztalásban:

```
400 print"Kedves "b$"! "
410 print"Igazán tudhatnál már ennyit!"
420 e=(18-val(c$))
430 print"Alig "e" év múlva érettségizel,"
440 print" addigra remélem megtanulod!"
```

A válaszelemző modul kivitelezésében az alábbiakra figyeljünk különösen:

- Számadat fogadása, számolási pontosság
- Áttekinthetőség (IF...THEN-sorok)
- Minden elképzelhető válasz feldolgozása
- Szubrutinok alkalmazása
- Hibaregisztrálás, ékezethibák
- Értékelés részletesen és összegezve
- Stílus

8. Grafika

Grafikai szempontból a mikroszámítógép intelligens és interaktív rajzeszközként fogható fel. Intelligens, amennyiben a rajzhoz szükséges adatokat saját maga számíthatja ki (pl. függvények ábrázolása). Interaktív, mert a felhasználóval folytatott párbeszéd alapján végezheti a rajzolást, rajzok mozgatását. Az esetek legnagyobb részében – főleg iskolákban – a rajzeszköz kifejezést idézőjelben kell érteni, hiszen csak a képernyőre tud rajzolni, nem lévén nyomtató. Ha lassan is, de terjedni fog a nyomtatók iskolai felhasználása – az egyszerűbb mátrixnyomtatók ára két-háromszorosa a mikroszámítógépek árainak – szolgáltatásaikat érzékelteti, hogy e könyv ábráinak nagy része is ilyen nyomtatón készült.

A számítógép elsődleges outputja (kimenete) a képernyő, amely egyszínű, homogén keretből és az információt hordozó pontrácsból áll. E pontrács a PRIMO gépeknél 192x256, a Commodore gépeknél 200x320, a TVC-nél 240x512 pontból áll. A képernyőn elhelyezhető karakterek a pontrács 8x8 (Commodore),6x12 (PRIMO), illetve 10x8 (TVC) pontját foglalják el, ezzel 25 sort és 40 oszlopot (Commodore), 16 sort, 42 oszlopot (PRIMO), illetve a TVC-n maximum 24 sort, 64 oszlopot hozva létre.

A PRIMO és TVC gépeknél karakterek megjelenitése mellett a pontrács bármely pontját is kivilágíthatjuk, azaz rajzolhatunk. A Commodore gépek két alapvető üzemmóddal rendelkeznek:

- karakteres, vagy alacsony felbontású,
- bittérképes, vagy nagyfelbontású.

A karakteres üzemmódban a karakter ROM-ban tárolt, vagy definiált 256 karakter bármelyike a képernyő 25 sorának 40-40 helyére vihető.

A bittérképes üzemmódban a 200x320-as képernyő minden pontját külön-külön lehet bekapcsolni és bizonyos korlátozásokkal szinezni. E sok pont adatainak tárolása természetesen jelentősen csökkenti a szabad tárkapacitást.

A számítógép képernyőjén lévő ábrák mozgatása a rajzfilmhez hasonló elven alapszik: a képek gyors egymásutánban változnak, igy szemünk folyamatosnak látja. Ez a megállapitás azonban csak a gépi kódban megirt programokra igaz maradéktalanul, BASIC-programból lassúbb a mozgatás. A BASIC-interpreter soronként fordítja, értelmezi a programot – ez nagyságrendekkel lassúbb mozgást eredményez, mint pl. a játékprogramokban látottak, vagy a képzett programozók által készitett – igen kevés(!) – grafikus oktató program némely megoldása.

Mielőtt mozgásos grafikus program készitéséhez fognánk, dönteni kell az interaktivitással kapcsolatos lehetőségekben. Egy mozgást a felhasználó által nem befolyásolható módon bemutató program helyett jobb az animációs film – más kérdés, hogy programot, vagy rajzfilmet könnyebb-e készíteni. Szerző mindkettőt kipróbálva nyugodt lelkiismerettel ajánlhatja nem interaktiv megoldáshoz a teljes formai és mozgatási szabadsággal biró animációs filmet. A mikroszámítógép grafikai lehetőségeit sok tényező korlátozza, amelyeket még a legképzettebb hivatásos programozók verejtékes munkájával készült népszerű játékprogramokban is felfedezhetünk.

A korlátok után nézzük a lehetőségeket. Az ábrák mozgását vezérelheti a program, vagy közvetlenül a felhasználó. Program vezérelte mozgatásnál az interaktivitás a mozgatás megkezdése előtt a mozgás módjára, irányára, sebességére, stb.-re adott felhasználói

kívánságok alapján valósulhat meg. Magasabb szintű az interaktivitás, ha a felhasználó programfutás közben joystickkel, vagy kurzormozgató billentyűkkel irányithat valamilyen mozgást. A kétféle vezérlés természetesen kombinálható is.

A következőkben először az egyszerűbb karakteres grafikával, majd a bonyolultabb, de többet nyujtó bittérképes grafikával foglalkozunk.

8.1 Karakteres grafika

8.1.1. Rajzolás

A Commodore gépek sok grafikus karakterrel rendelkeznek, de bármelyik gépen lehetséges megfelelő karakterek definiálása - akár grafikus célra is. Figyelembe kell azonban venni, hogy BASIC programmal a karaktereket csak a soroknak és oszlopoknak megfelelően tudjuk a képernyőn elhelyezni. Emiatt a karakterek mozgatása kissé "darabos", ugráló.

A Commodore gépek igen jól összeállitott grafikus karakterkészletével (62 karakter!) sok műszaki, technológiai jellegű ábra felépithető. A grafikus karakterek megfelelő cserélgetésével mozgást, folyamatot is lehet szimulálni. Az alábbi C64-re készült program két tartályt ábrázol, amint az egyik tartályból fogyó anyag a másik tartályban átalakulva jelenik meg:

```
20 poke1041,100:poke1042,100
30 poke1080,103:poke1083,101
40 poke1120,103:poke1123,101
50 poke1160,103:poke1163,101
60 poke1201,99:poke1202,99:poke1203,9
65 poke1204,76:poke1205,111:poke1206,111
70 poke1244,103:poke1247,101
80 poke1284,103:poke1287,101
90 poke1324,103:poke1327,101
100 poke1365,99:poke1366,99
110 poke1081,99:poke1082,99:gosub1000
120 poke1081,69:poke1082,69:gosub1000
130 poke1081,67:poke1082,67:gosub1000
140 poke1325,104:poke1326,104
145 poke55597,7:poke55598,7:gosub1000
150 poke1081,82:poke1082,82:gosub1000
160 poke1081,32:poke1082,32:poke1121,99
165 poke1122,99:gosub1000
170 poke1121,69:poke1122,69:poke1325,102
175 poke1326,102:gosub1000
180 poke1121,67:poke1122,67:gosub1000
190 poke1121,82:poke1122,82:poke1285,104:poke1286,104
195 poke55557,7:poke55558,7:gosub1000
200 poke1121,32:poke1122,32:poke1161,99
205 poke1162,99:gosub1000
210 poke1161,69:poke1162,69:poke1285,102
215 poke1286,102:gosub1000
220 poke1161,67:poke1162,67:gosub1000
230 poke1161,82:poke1162,82:gosub1000
240 poke1245,104:poke1246,104:poke55517,7:poke55518,7
```

```
990 stop
1000 fora=1 to 300:next:return
```

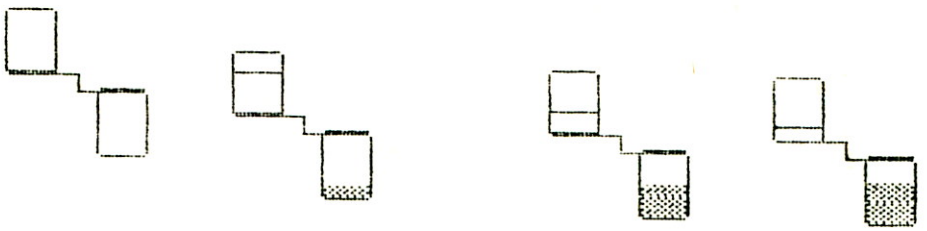
A program POKE utasításokkal közvetlenül a képernyőtárba viszi be a karakterkódokat:

20 - 100 sorok: a két tartály megrajzolása
110 - 130 sorok: felső tartály kezd ürülni
140 : alsó tartályban megjelenik a termék
1000 : lassító ciklus szubrutinja

A szín megváltoztatását az alsó tartályban szintén POKE utasítással végezzük: a karakter bevitele után a szintár megfelelő helyeire beírt színkóddal (140, 190, 240 sorok). A felső tartály folyadékszintjének csökkenését egy karakterhelyen belül négy karakter cseréjével (képernyő kódjuk: 99-69-67-82) végezzük. Egy sorral lefelé lépéskor az addigi karkterhelyen lévő "folyadékszintet" a SPACE billentyű kódjának (32) beírásával törölhetjük (160,200 sorok). Minden változás után lassító ciklusra van szükség (GOSUB 1000), különben láthatatlanul gyors lesz a folyamat.

Az ilyen programot négyzetrácsos papíron előre pontosan meg kell tervezni, a négyzetrácsokba beírva a képernyőtár címeit, s csak ezután írni be a gépbe a sok POKE-utasítást, illetve DATA-sorokat (ld. alább). Az első pillanatra ijesztően sok számot tartalmazó lista megtévesztő: a szerző rövid óra alatt írta ezt a programot a számolással együtt! A négyzetrácson lévő rajzot el kell tenni, így más Commodore gépre való átíráskor csak a címeket kell változtatni: a Plus/4 képernyőtár kezdőcíme: 3072, a C64-é: 1024. Plus/4-re való átíráshoz tehát a POKE utasítások első paraméterét kell 3072-1024=2048-al megnövelni. A szintárok kezdőcímei: C64: 55296, Plus/4:2048.

A 990-es sor azért szükséges, mert különben az 1000-es sorra jutva a program egy GOSUB (szubrutin-hívás) nélküli RETURN utasítást találva hibát jelez (STOP helyett END is lehet). Az alábbi ábrák mutatják a programfutás néhány fázisának képernyőjét.



Vannak olyan programok, melyekben igen sokszor szerepel ugyanaz az utasítás, mint fentiben a POKE. Ilyenkor a DATA-READ utasításpárral célszerű ciklust szervezni az adatok beolvasására. Kezdő programozó könnyebben megírja a fenti listát, de érdemes törekedni a programok rövidítésére, tömörítésére, ezáltal gyorsabb is lesz a programfutás. Természetesen az ilyen egyszerű programnál nincs jelentősége a rövidítésnek és gyorsításnak, de a programfejlesztés folyamata miatt nézzük meg az alábbi listát, amelyik pontosan a fenti programmal megegyező eredményt ad.

```
10 print"S"
20 read a,b
30 if a=0 and b=0 then gosub 1000:goto20
35 if a=-1 then end
40 poke a,b: goto 20
50 data 1041,100,1042,100,1080,103,1083,101
```

```
60 data 1120,103,1123,101,1160,103,1163,101
70 data 1201,99,1202,99,1203,99,1204,76
75 data 1205,111,1206,111
80 data 1244,103,1247,101,1284,103,1287,101
90 data1324,103,1327,101,1365,99,1366,99
100 data 1081,99,1082,99,0,0
110 data 1081,69,1082,69,0,0
120 data 1081,67,1082,67,0,0
130 data 1325,104,1326,104,55597,7,55598,7,0,0
140 data 1081,82,1082,82,0,0
150 data 1081,32,1082,32,1121,99,1122,99,0,0
160 data 1121,69,1122,69,1325,102,1326,102,0,0
170 data 1121,67,1122,67,0,0
180 data 1121,82,1122,82,1285,104,1286,104
185 data 55557,7,55558,7,0,0
190 data 1121,32,1122,32,1161,99,1162,99,0,0
200 data 1161,69,1162,69,1285,102,1286,102,0,0
210 data 1161,67,1162,67,0,0
220 data 1161,82,1162,82,0,0
230 data 1245,104,1246,104,55517,7,55518,7,-1,-1
1000 for x=1 to 300:next x:return
```

A 20-40 sorok ciklusa a DATA sorokból beolvassa A és B értékét, a 40-es sor POKE A,B utasítása pedig végrehajtja a rajzolást. A 30-as sor a lassító ciklusokat biztosítja a DATA sorokba írt (0,0) adatoknak megfelelő helyeken. A 40-es sor észleli az adatsor végét.

A két lista, vagy a programfutás összehasonlítása nem mutat lényeges különbséget a két megoldás között, de hosszabb, bonyolultabb programoknál már jelentős különbségek mutatkoznak, ezért ahol lehet ilyen ciklusszervezésekre kell törekedni.

A TVC gépek is rendelkeznek néhány grafikus karakterrel , de könnyen szerkeszthetünk tetszőlegeseket, például a Commodore gépeken megszokottakat.

8.1.2. Mozgatás

A karakterek mozgatását lehetővé teszi, hogy a POKE utasításokban a paraméterek számok helyett változók is lehetnek. Ciklusszervezéssel folyamatosan változtatva a POKE első paraméterét, a legegyszerűbb mozgatási lehetőséghez jutunk:

```
10 print"<CLR>"
20 input"Sebesség:";b
30 for a= 1504 to 1543
40 poke a-1,32:poke a,81
50 for i= 1 to 1000 step b
60 next i
70 next a
```

Ebben a kis programban az [a] változó a C64 egy képernyősorának kezdőcímétől a végéig változik egyesével. A cikluson (70-30 sor) belül POKE utasításokkal az aktuális [a] értékkel megadott címre elhelyezzük a mozgatott karaktert (labda, képernyőkód: 81), de előtte [a-1] címen egy SPACE-el töröljük az előzőleg odahelyezett karaktert. Az 50-es sor végzi a lassítást: a felhasználó által választható [b] érték arányos a sebességgel.

Lehet több karaktert egyszerre mozgatni, de egymásba ágyazott ciklusok estén is egyszerre csak egy FOR...NEXT ciklus van érvényben. Ezért, ha különböző irányú mozgásokat szeretnénk, akkor egy változóból kell matematikai műveletekkel létrehoznunk a többféle mozgást:

```
10 for a= 1 to 20
20 poke 3111+2a,32:poke 3112+2a,81
30 poke 3191-2a,32:poke 3190-2a,81
40 poke 3090+a*40,32:poke3130+a*40,81
50 poke 3072+a*41,32:poke3113+a*41,81
60 next a
```

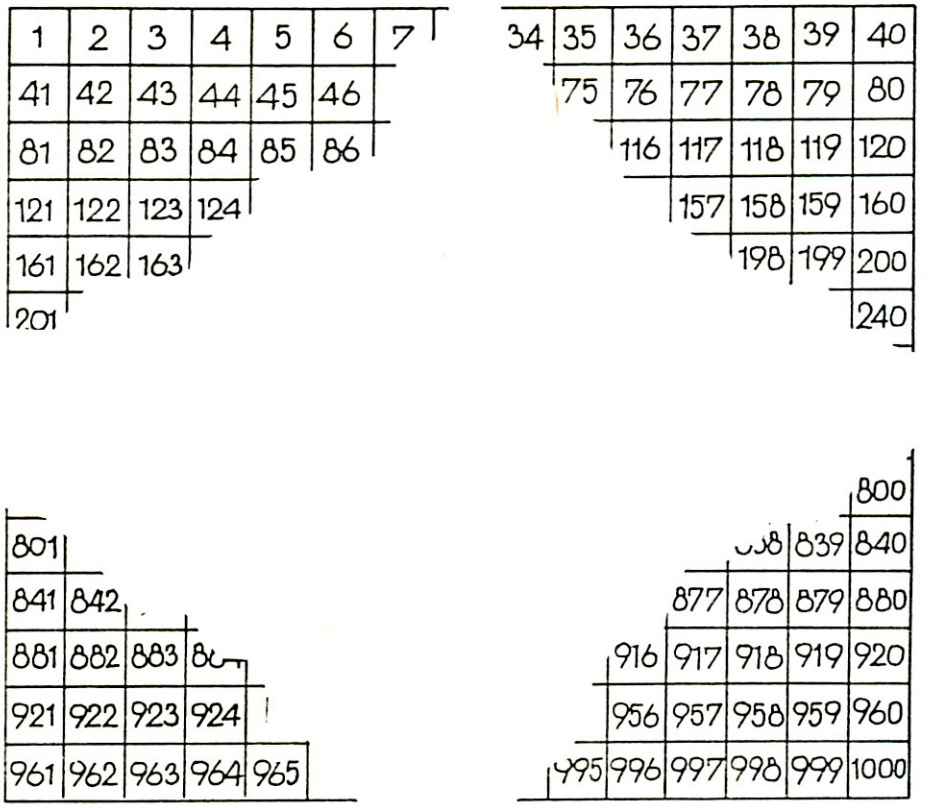
E kis program (C16, Plus/4) 20-as sora vízszintesen jobbra, 30-as sora vízszintesen balra, 40-es sora függőlegesen lefelé és 50-es sora átlósan mozgatja a labdát. Az ilyen mozgatásoknál a törlési címre kell jobban odafigyelni, néhány próbálkozás gyakran szükséges a sikerhez.

A több mozgatott karakter össze is kapcsolható, így mozgó alakzatok hozhatók létre. Ilyen esetben azonban eléggé villódzó képet kapunk, mert a BASIC program sebessége olyan, hogy több karakter törlése, újrarajzolása már szemmel észlelhető időcsúszást jelent (gépi kódú program sebessége nagyságrendekkel több, ezért nem villódzik). Az alábbi lista egy autó alakját mozgató program C64-re, de az 5-ös sorban FOR A=3272 TO 3302 ciklushatárokkal C16 és Plus/4 gépen is fut:

```
1 input"sebesseg=";b
5 fora=1224 to 1254 step b
10 for t=1 to 200:next t:print"S"
20 poke a+2,47:pokea+3,99:pokea+4,99
25 pokea+5,99:pokea+6,77
30 pokea+40,85:pokea+41,67:pokea+42,125
40 pokea+47,99:pokea+48,99:pokea+49,77
50 pokea+80,93:pokea+89,106
60 pokea+120,109:pokea+121,87:pokea+122,67
65 pokea+123,67:pokea+124,67
70 pokea+125,67:pokea+126,67:pokea+127,67
80 pokea+128,87:pokea+129,125
90 next a
```

Itt a törlés legegyszerűbb módját választottuk: a 10-es sor a ciklusváltozó minden növelésekor a rajzolás előtt törli az egész képernyőt. Ilyen sok karakter mozgatásánál a munkaigényesebb karakterenkénti törlés sem ad villódzásmentes képet. A számok írásának egyszerűsítése és a hibázási valószínűség csökkentése érdekében a négyjegyű számok (képernyőtár címek) helyett egybetűs változót érdemes használni, még akkor is, ha ez nem ciklusváltozó, mint fent. A rajzok négyzetrácsos papíron való megtervezését

könnyítheti az alábbi módszer:



Igy könnyebb a címeket a koordinátarendszerből kiolvasni, a program elején pedig beírjuk:

10 A=1023 (C64) , vagy
10 A=3073 (C16 és Plus/4)

Ekkor POKE a+1,81 utasítás az első sor első karakterhelyére írja a labdát.

A mozgatási sebességet az eddigi programokban kétféleképpen kértük be a felhasználótól:

- a ciklusváltozó növelésében a lépésköz megadásával:

```
10 input"sebesség:";b
20 for a=1 to 100 step b
```

- a lassító ciklusbeli lépésköz megadásával:

```
10 input"sebesség:";b
100 for i=1 to 1000 step b
```

Ez utóbbi esetben kérdezhetnénk azt is, meddig növekedjen a lassító ciklus változója, de ez nem szerencsés megoldás, hiszen így a nagyobb szám jelentené a lassúbb mozgást.

E sebességadatok bekérésénél – mint minden hasonló esetben – gondoskodni kell a nem megfelelő adatok kiszűréséről.Ezért itt is javasolt a sztringkénti beolvasás, majd numerikus változóvá alakítás. A lehetséges határértékeket érdemes előre közölni, és utólag is ellenőrizni:

```
10 print"A mozgatás sebessége 1 és 10 közötti"
20 print"érték lehet, válassz!"
30 input"sebesség:";a$
40 a=val(a$)
50 if a<1 or a>10 then 70
60 goto 100
70 print"Kérlek, 1 és 10 között válassz!": goto 30
```

A viszonylag egyszerű karakteres grafika sokféle szimulációs program elkészítésére ad lehetőséget. A következő program arra mutat példát, hogy a csúcstechnológia bonyolult eljárásainak lényegét is be lehet mutatni egyszerűen. A program egy ipari robot működését szimulálja: a kurzormozgató billentyűk felhasználásával "betanítjuk" a számítógépet, hogy hova tegyen a képernyőn csillagokat, utána akárhányszor ismétli a kézzel vezérelt folyamatot. A képernyőt nevezzük ki munkadarabnak, a csillagokat furatoknak, vagy hegesztési pontoknak – kész a számítógép vezérelte robot. De elképzelhetjük rajzgépnek is, amely az általunk szabadkézzel rajzolt "alkotást" ismételgeti kérésre. Ezzel a példával arra is akarunk utalni, hogy a szimuláció sokszor lehet olyan általános, ami csak a "csupasz" lényeget mutatja, de könnyen felruházható többféle konkrét "öltözékkel". Jó pedagógiai gyakorlattal és érzékkel eltalálható az optimális absztrakciós szint. A program listája (C16):

```
10  poke 1344,64
20  clr:print"<CLR>"
30  print"Hány elem legyen a művelet?"
40  input h$:h=val(h$)
50  dim a(h)
60  print"Indulhat!"
70  for i=1 to h
80  getkey a$
90  a(i)=asc(a$)
100 print chr$(a(i))"<CRSR jobbra>*";
110 next i
120 print:print:print"Gombnyomásra indul."
130 getkey t$: print"<CLR>"
140 for i=1 to h
150 print chr$(a(i))"<CRSR jobbra>*";
160 for z=1 to 100: next z
170 next i
180 print:print"Ismételjem? i/n"
185 print"Újra kezdjük? u"
190 getkey i$
200 if i$="i" then print"<CLR>":goto 140
210 if i$="n" then end
215 if i$="u" then 20
220 print"Kérlek i, n, vagy u betűvel válaszolj!":
goto 180
```

A program 10-es sora letiltja a billentyűismétlést, a 20-as sorban CLR törli a változó tömböt, hogy újra definiálhassuk 50-ben. A 90-es sor a billentyűkód értékét indexes numerikus változóvá alakítja, melynek értékei a ciklus ismételgetésekor töltik fel a lefoglalt tömböt. A 150-es sor minden lenyomásnál kirajzolja a kívánt helyre a csillagot. Amikor a ciklusváltozó eléri a bekért [h] értékét, a program a 130-as sorban várakozik, majd gombnyomásra újabb ciklusban a 150-es sor a változó értékének megfelelő karaktert állítja elő (kurzormozgató vezérlő karakter) és egy csillagot nyomtat. A 160-as

sor a már megszokott lassító ciklus. Természetesen a program felhasználásakor megfelelő szöveget is kell elébe írni, közölni, hogy csak a kurzormozgató billentyűket használja. Ki is lehet szűrni más billentyűk használatát az alábbi részlet betoldásával:

```
82 if a$="<CRSR balra>" or "<CRSRjobbra>" or "<CRSR le>" or "<CRSR fel>"
then 90
84 print"Csak a kurzormozgató billentyűket használd!": goto 80
```

Az interaktivitás magasabb szintjét valósítjuk meg – és érdekesebb programot készíthetünk – ha a mozgás a felhasználó által vezérelhető. A kurzormozgató billentyűk használatának előnye, hogy semmi egyéb hardver-eszközt nem igényel. A joystickkel való vezérlés hátránya, hogy joystick nélkül nem működik, előnye viszont a 8 irányba való haladás, és a gyerekek szivesebben nyomkodják a botkormányt. A joystick említett hátrányát könnyű kiküszöbölni : a program elején megkérdezni, hogy a felhasználó mivel kíván irányítani, s ez alapján elágaztatni a mindkettőt tartalmazó programot.

Mindkét megoldás alapja egy ciklus, amelyik állandóan ellenőrzi a billentyűzetet, illetve a joystick állását, és ez alapján kiszámítja a mozgó karakter aktuális koordinátáit. Az alábbi program C16-on mozgat egy csillagot:

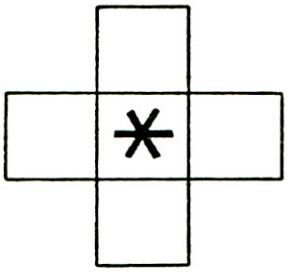
```
10 rem"csillag mozg."
20 u=20:v=12: goto 130
30 getkey a$
40 i=asc(a$)
50 if i=29 then u=u+1
60 if i=157 then u=u-1
70 if i=17 then v=v+1
80 if i=145 then v=v-1
90 if u>37 then u=u-1
100 if u<3  then u=u+1
110 if v>22 then v=v-1
120 if v<2  then v=v+1
130 char 1,u,v," * <CRSR le><CRSR bal><CRSR bal> <CRSR fel><CRSR fel>
<CRSR bal> ":goto 30
```

A 20-as sor megadja a kezdő koordinátákat (képernyő közepe), majd a 50-80 sorok végzik a koordináták átírását. A 40-es sor a lenyomott billentyű ASCII-kódját adja, ami a Commodore gépeknél megegyezik:

billentyű	ASCII-kód
kurzor fel	145
kurzor le	17
kurzor jobbra	29
kurzor balra	157

A 90-120-as sorokban korlátot szabunk a mozgatásnak, mert csak a képernyő oszlop- és sorszámának engedélyezett tartományába (0-39, 0-24) eshetnek a CHAR utasítás paraméterei. A rajzolást és a törlést a 130-as sor végzi. A CHAR utasítás sztringje tartalmazhat vezérlő karaktereket is, így a csillag előtt, mögött, alatt és felett

helyeztünk el egy-egy üres szóközt (SPACE):



Ezáltal bármerről is érkezik a csillag az adott helyre, mind a négy lehetséges megelőző pozíciót töröljük. A ciklus állandóan működik, STOP-al lehet leállítani. Egyszerű megoldani, hogy valamelyik billentyű megnyomására álljon meg:

```
85 if i=32 then end
```

Ekkor a SPACE-billentyűvel leállítható a program.

Lényeges ellenőrizni a működő programot abból a szempontból, hogy valamilyen nem kívánatos billentyű lenyomása zavart okozhat-e. Ebben a programban az IF...THEN sorok csak a kurzormozgató billentyűkre vizsgálnak, tehát bármi mást nyomunk meg, továbbhalad a program: a 130-as sor újra kirajzolja az előbbi helyre a csillagot, majd kezdődik előlről a ciklus - vagyis semmit nem észlelünk.

A joystickel való mozgatásnál a ciklusnak a JOY(1), illetve JOY(2) függvény (1-es, vagy 2-es joystick) értékét kell figyelni a C16 és Plus/4 gépeknél. E függvény értékei:

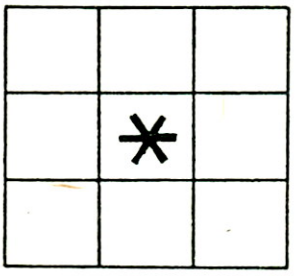
```
                FENN
                  1
              8       2
BAL       7       0       3       JOBB
              6       4
                  5
                LENN
```

Amennyiben a tűzgombot is lenyomjuk, a fenti értékek 128-al nagyobbak. Az alábbi program a csillag mozgatását végzi:

```
140 u=20:v=12:goto 270
150 if joy(1)=3  thenu=u+1
160 if joy(1)=7  thenu=u-1
170 if joy(1)=5  then v=v+1
180 if joy(1)=1  then v=v-1
190 if joy(1)=4  then u=u+1:v=v+1
200 if joy(1)=2  then u=u+1:v=v-1
210 if joy(1)=8  then u=u-1:v=v-1
220 if joy(1)=6  then u=u-1:v=v+1
230 if u>37 then u=u-1
240 if u<3  then u=u+1
250 if v>22 then v=v-1
260 if v<2  then v=v+1
270 char 1,u,v," *  qiiiii     QQiiiii     ":goto150
```

A 270-es sorban a betűk jelentése:
q:<CRSR le>, Q:<CRSR fel>, í:<CRSR balra>

A 270-es sor itt hosszabb, mert 8 lehetséges irányból érkezhet a csillag éppen aktuális helyére:



A sok IF...THEN sor lassítja program futását, ezért célszerű ON-al helyettesíteni:

```
150 on joy(1) goto 160,170,180,190,200,210,220,225
160 v=v-1:goto 230
170 u=u+1:v=v-1:goto 230
180 u=u+1:goto 230
190 u=u+1:v=v+1:goto 230
200 v=v+1:goto 230
210 u=u-1:v=v+1:goto 230
220 u=u-1:goto 230
225 u=u-1:v=v-1:goto 230
```

Így egy sorral hosszabb lett a program, de gyorsabb, mert nem kell az összes IF-sort vizsgálni minden ciklusban. A 150-es sor JOY(1) értéke alapján közvetlenül a megfelelő sorba irányít.

A Commodore 64-es gépen valamivel nehezebb a fentiek megoldása, mert nem ismer a CHAR utasításnak és a JOY függvénynek megfelelőt. A joystick pillanatnyi állását az 56320 (1-es joystick), illetve 56321 (2-es joystick) címek tartalmazzák. A PEEK(56320) értékei:

```
                  FENN
                  126
            122         118
BAl   123                     119   JOBB
            121         117
                  125
                  LENN
```

Tűzgomb lenyomására 16-al csökkennek a fenti értékek.

Az alábbi kis program a C64-en az előző programhoz hasonlóan működik. A törlést és kirajzolást itt a POKE utasítással lehet végezni, aminek egy szépséghibája van: sor végére érkezve az írásnak megfelelően a képernyő másik szélén folytatja mozgását.

```
20 b=1444:goto 130
40 a=peek(56320)
50 if a=119 then b=b+1
60 if a=118 then b=b-39
```

```
70 if a=123 then b=b-1
75 if a=122 then b=b-41
80 if a=125 then b=b+40
85 if a=117 then b=b+41
90 if a=121 then b=b+39
100 if a=126 then b=b-40
110 if b<1125 then b=b+40
120 if b>1922 then b=b-40
130 pokeb,81:pokeb-1,32:pokeb+1,32
140 pokeb-41,32:pokeb-40,32:pokeb-39,32
150 pokeb+39,32:pokeb+40,32:pokeb+41,32
160 goto 40
```

Itt is meg lehet akadályozni a nem kívánatos értékeket. A képernyő első és utolsó címén való túljutást feltétlenül tiltani kell (110- és 120-as sorok), mert ellenőrizhetetlenül átírhatjuk a POKE-utasítasokkal az egész memóriát. A sor végi továbbhaladás megakadályozása valamivel nehezebb. Az összes sor végi címet (48 cím!) fel kell sorolni, és ezekre az esetekre megtiltani a rajzolást. Szerencsére ciklussal meg lehet úszni sokkal rövidebben:

```
122 FOR J=1062 TO 2022 STEP 40
124 IF ABS(B-J)<0.1 THEN B=B-1
125 NEXT J
126 FOR BA=1025 to 1985 STEP 40
128 IF ABS(BA-B)<0.1 THEN B=B+1
129 NEXT BA
```

Ezzel a megoldással ugyan lelassítjuk a programot, de a kép bal, vagy jobb szélét egy betűköznyire megközelítve nem engedi a rajzolást, hanem megállítja.

A TVC gépeken hasonlóan írható meg ez a program, a CHAR utasítást a PRINT AT-vel helyettesítve:

```
500 rem"csillag mozg."
505 cls:graphics4
510 u=10:v=20:goto 650
520 a$=inkey$:if a$="" then 520
530 if a$=chr$(5) then u=u-1:goto 600
540 if a$=chr$(24) then u=u+1:goto 600
550 if a$=chr$(4) then v=v+1:goto 600
560 if a$=chr$(19) then v=v-1:goto 600
600 if u>22 then u=22
610 if u<1 then u=1
620 if v>30 then v=30
630 if <1 then v=1
650 print at u,v:" * "
660 print at u-1,v:"   "
670 print at u+1,v:"   ":goto 520
680 end
```

Már említettük, hogy a sok IF...THEN sor lassítja a program futását. A C16 és Plus/4 gépek BASIC-je ismeri az INSTR függvényt:

```
INSTR (<sztring 1>,<sztring 2>)
```

értéke megadja, hogy a <sztring 2> a <sztring 1>-nek hányadik karakterén kezdődik (ha <sztring 2> nem része <sztring 1>-nek, akkor 0 az értéke). A csillagot kurzormozgatókkal vezérlő programot helyettesíthetjük az alábbival a 130-as sorig:

```
10 a$=chr$(145)+chr$(29)+chr$(17)+chr$(157)
20 u=20:v=12:goto 130
30 getkey b$
40 on instr(a$,b$)+1 goto30,50,60,70,80
50 if u>1 then u=u-1
55 goto 130
60 if v<37 then v=v+1
65 goto 130
70 if u<23 then u=u+1
75 goto 130
80 if v>0 then v=v-1
85 goto 130
```

Az a$ sztring a kurzormozgató billentyűk összesét tartalmazza, és a 40-es sor aszerint irányít, hogy a$ értéke hányadik karakterének megfelelő billentyűt nyomtuk meg. A koordináták új értékeit megadó sorokban egyúttal ki is szűrjük a nem megengedett értékeket (eredetiben 50-120 sorok). Ez a program azt a lehetőséget villantja fel, hogy apró fogásokkal lehet fejleszteni, gyorsítani egy programot - a konkrét esetben elég gyors az első megoldás is.

A karakteres grafika könnyen megismerhető, és jól használható programokra ad lehetőséget. A karakterek megfelelő módosítását elvégezve tetszés szerinti mozgatásokat tartalmazó szimulációs és játékprogramok készíthetők. A karakteres grafika nagy előnye a pontgrafikával szemben gyorsasága: BASIC programmal is megfelelő sebességű mozgásokat hozhatunk létre.

8.2. Pontgrafika

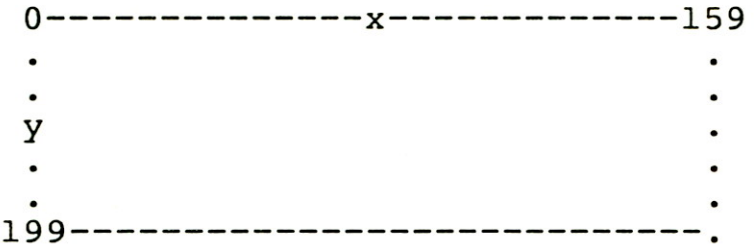
A nagyfelbontású, vagy bittérképes grafikát legegyszerűbben - magyarosan - pontgrafikának lehet nevezni, hiszen a teljes képet pontokból kell megalkotni. A Commodore gépeken ennek az üzemmódnak két fajtáját használhatjuk:

1. Standard (nagyfelbontású) bittérképes mód
2. Többszínű bittérképes mód

Az elsőben a képernyő 200x320 pontot tartalmaz az alábbi koordinátarendszer szerint:

Ebben az üzemmódban csak két szint lehet használni, a bittérkép tárolása még így is 8 kilobyte helyet foglal.

A többszínű bittérképes üzemmód négy szín használatát teszi lehetővé annak az árán, hogy a vízszintes felbontás felére csökken. A megfelelő koordinátarendszer:



A Commodore 64 BASIC-je (BASIC V2) sajnos nem tartalmaz grafikus utasításokat, így a grafikus üzemmód használata segédprogram nélkül igen nehéz, nem kezdő programozóknak való feladat. A 8.2.3. fejezetben kitérünk az ilyen segédprogramokra és az úgynevezett sprite-üzemmódra, amely a C64 egyedi lehetősége, viszonylag könnyen programozható.

A C16 és a Plus/4 BASIC-je (BASIC V3.5) közvetlenül támogatja a grafikus üzemmódok használatát. E gépek grafikus üzemmódban lehetőséget adnak az osztott képernyőre: a grafikus képernyő alsó ötödét karakteresen használhatjuk. Ez a lehetőség oktatóprogramoknál nagyszerűen használható a rajz melletti közlésekre, kérdésekre a megszokott utasításokkal (PRINT, INPUT, GET, stb.). Az üzemmódok kiválasztását a GRAPHIC a,b formájú utasítással végezhetjük. Az első [a] paraméter 0-4 között lehet, jelentése:

0: karakteres üzemmód
1: nagyfelbontású grafika
2: nagyfelbontású grafika, osztott képernyő
3: többszínű grafika
4: többszínű grafika, osztott képernyő

A második [b] paraméter 0 vagy értékű lehet: ha 1, akkor törli az előző grafikus képernyő tartalmát, ha 0,akkor az előző tartalommal jeleníti meg. Amennyiben nem írunk második paramétert, akkor 0-nak veszi, vagyis megmarad az előző tartalom. A grafikus képernyő törlése nem lehetséges a [CLR] gombbal, vagy a PRINT"<CLR>" utasítással - ezek csak a karakteres képernyőt törlik, osztott képernyőnél a karakteres alsó részt. A grafikus képet törli a memóriából a GRAPHIC CLR utasítás (parancs is lehet), míg az SCNCLR törli az üzemmódnak megfelelő teljes képernyőt (grafikus és karakteres részt is).

Jól használható programban a GRAPHIC 0 utasítás, amely törli a grafikus képernyőt, de a megfelelő memóriát nem, azaz később a GRAPHIC utasítás második paraméter nélküli (vagy =0) alakjával visszahozható az előző grafikus képernyőtartalom.

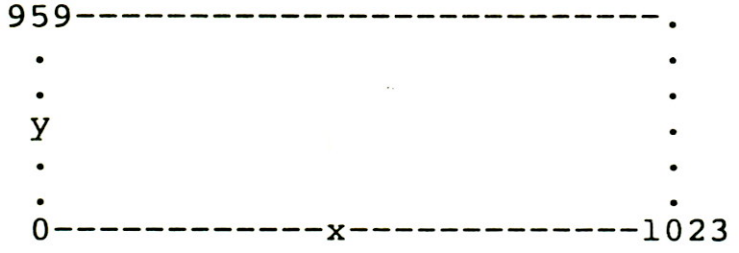
A PRIMO gépek egy üzemmódot használnak: egyszerre megjeleníthetők karakterek és grafikus ábrák, rajzok minden korlátozás nélkül. A grafikus funkciót két utasítás valósítja meg:

SET(x,y) : a képernyő x,y koordinátájú pontját bekapcsolja
RESET(x,y): az x,y koordinátájú pontot kikapcsolja

A PRIMO képernyőjének megfelelő koordinátarendszer:

A koordinátarendszer ilyen beosztása jobban megfelel az iskolában tanultaknak. Más grafikus utasításai nincsnek a PRIMO BASIC-jének, ezért a rajzolás hosszadalmasabb, mint a C16-on. A PRIMO programokban bizonyos rajz-elemeket (kör, egyenes,stb) elkészítő rutinokkal lehet pótolni a C16 grafikus utasításait.

A TVC gépeknek három üzemmódja van, mindháromban karakterek és rajzok is megjeleníthetők. A koordinátarendszer beosztása olyan, hogy mindegyik üzemmódban azonosan használhatjuk, a gép átszámítja :



Az üzemmódok a használható színek száma szerint:

1. Kétszínű üzemmód.
   Beállítása a GRAPHICS 2 utasítással történik. Ekkor 24 sorban, soronként 64 karakter jeleníthető meg, illetve 240x512 képpont címezhető, a függőleges koordináták 4-el, a vízszintesek 2-vel osztódnak. Ekkor csak két szín (háttér és írás) használható.

2. Négyszinű üzemmód.
   Beállitása a GRAPHICS 4 utasitással lehetséges: 24 sorban, soronként 32 karakterhelyet, illetve 240x256 képpontot tartalmaz a képernyő. A koordináták osztója függőlegesen és vizszintesen is 4.
3. Tizenhatszinű üzemmód.
   Beállitása: GRAPHICS 16. Ekkor 24 sorban soronként 16 karakter, illetve 240x128 képpont jeleníthető meg. A koordináták osztója függőlegesen 4, vizszintesen 8.

A TVC rendszerének az a nagy előnye, hogy a koordináták értékeit bármely üzemmódban azonosan adhatjuk meg, pl. a képernyő középpontja mindig az (511, 479) koordinátapárral adható meg. A grafikus utasitáskészlet a PRIMO gépekéhez hasonló: A PLOT utasitassal lehet pontokat, illetve vonalakat rajzolni, minden egyéb rajzot e pontokbol és vonalakból kell felépiteni.

8.2.1. Rajzolás

Rajzolás szempontjából a PRIMO utasitáskészlete a legegyszerűbb: SET, RESET. Pontokat elhelyezni a SET utasitással könnyű, azonban minden ábrát pontokból kell felépiteni, s ez már munkaigényes. A SET és RESET utasitások paraméterei lehetnek változók, igy ügyes ciklusszervezésekkel jelentősen felgyorsithatjuk a vonalak (=pontsorok) rajzolását. Húzzunk egy vizszintes egyenest a képernyő közepén! Az egyesnesszakasz kezdőpontjának koordinátái: 0,95 , a végpont koordinátái: 255,95. Láthatjuk, hogy az egyenesszakasz rajzolásához az x-koordinátát kell 0-tól 255-ig növelni, tehát a program:

```
10 for x=0 to 255
20 set(x,95)
30 next x
```

Ha nem vizszintes, vagy függőleges egyenest szeretnénk rajzolni, akkor nyilván mindkét paraméternek változnia kell. Egyszerre azonban csak egy ciklus lehet érvényben, tehát a labdamozgató programhoz hasonlóan matematikai művelettel kell egy ciklusváltozóból kialakitani a megfelelően változó koordinátákat. A bal felső sarokból induló átló programja:

```
10 for x=0 to 255
20 y= 191-x*191/255
30 set(x,y)
40 next x
```

Mivel itt az y-koordinátának a maximumtól kell csökkennie (felső sarok!) ezért a kivonás. x-koordinátából az y-koordinátát a képernyő méretarányának megfelelő szorzással kaphatjuk:

```
y = x*191/255        x = y*255/191
```

Ha a két koordinátát nem arányosan kivánjuk változtatni, akkor természetesen a szorzótényező más lesz. Cseréljük a 20-as sort az alábbira:

```
20 y= 191-x*191/510
```

Most x értéke az előzőnek csak fele, tehát y-koordináta értéke 191-től nem 0-ig, hanem 95-ig fog csökkenni, azaz a bal felső sarokból a jobb képernyőoldal közepéig tart az egyenesszakasz.

A matematikában kevéssé járatos olvasó számára leírunk egy olyan szubrutint, amely a programba írva bármikor meghívható egyenesszakaszok rajzolására. Az első sorokban megkérdezi a rajzolandó szakasz kezdő és végpontját, valamint a lépésközt (l=1:folyamatos vonal, l>1: szaggatott).

```
1000 input"kezdőpont x-koordináta:";x1
1010 input"kezdőpont y-koordináta:";y1
1020 input"végpont x-koordináta:";x2
1030 input"végpont y-koordináta:";y
1040 input"lépésköz:";l
1050 if x1=x2 and y1=y2 then print"ez pont!"
1060 dx=x1-x2:dy=y2-y1
1070 if abs(dx)>abs(dy) then lk=dy/dx else lk=dx/dy:goto 1110
1080 for x=x1 to x2 step l*sgn(dx)
1090 y=y1+(x-x1)*lk: set(x,y)
1100 next x: return
1110 for y=y1 to y2 step l*sgn(dy)
1120 x=x1+(y-y1)*lk: set(x,y)
1130 nexty:return
```

Ez a rutin ugyanúgy végzi a rajzolást, mint az előtte látott egyszerű példák, csak általánosan lett megfogalmazva.

Körök rajzolása nem nélkülözheti a polárkoordináták ismeretét. Ezért erre a célra is közlünk egy általános szubrutint, amely matematikai ismeretek nélkül, teljesen mechanikusan is nyugodtan használható:

```
10 print"A középpont koordinátái:"
20 input"u=";u
30 input"v=";v
40 input"kör sugara:";r
50 for a=0 to 6.28 step 1/r
60 x=u+r*sin(a)
70 y=u+r*cos(a)
80 set(x,y)
90 next a
```

A kör rajzolását a 50-90 sorok végzik. Ellipszis rajzolásához a koordináták arányait kell megváltoztatni. Vízszintesen elnyúlt ellipszist kapunk az x-koordináta u-tól jobban eltávolodó értékeivel, pl.:

```
60 x=u+2*r*sin(a)
```

Függőlegesen nyújtott ellipszishez az y-koordinátát kell hasonló módon változtatni:

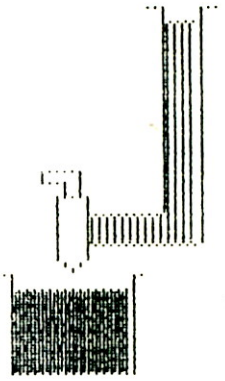
```
70 y=y+2*r*cos(a)
```

Körívet is rajzolhatunk az 50-es sor változtatásával: 0 - 6.28 a teljes kör, 0 - 3.14 félkört rajzol felülről indulva, az óramutató járásának irányába haladva, stb.

Besatírozott ábrák több egyenes egymás mellé rajzolásával készíthetők. Az alábbi programrészlet a "Titrálás" című programból egy büretta és az alatta lévő pohár rajzolását végzi el (mellette a rajz):

```
10 for p=238 to 247
20 for r=80 to 180
```

```
30 set(p,r)
40 next r
50 next p
60 reset(247,80):rem"sarok"
70 for p=225 to 238
80 for r=80 to 85
90 set(p,r)
100 next r
110 next p
120 for r=86 to 95
130 set(227,r):set(228,r)
140 next r
150 for p=220 to 226
160 set(p,94):set(p,95)
170 next p
180 for r=75 to 79
190 set(225,r):set(230,r):next r
200 for r=74 to 79
210 set(226,r):set(229,r):next r
220 for r=72 to 79
230 set(227,r):set(228,r):next r
240 for r=50 to 72
250 set(215,r):set(240,r):next r
260 for p=216 to 239
270 for r=49 to 57
280 set(p,r)
290 next r
300 next p
310 end
```



A 10-es és 20-as sor két ciklusváltozót ad, ezek alapján rajzol a 30-as sor. Természetesen egyszerre csak egy ciklus fut: először a belső ciklus (r), majd a külső ciklus (p) - igy a kitöltött büretta egymás mellé rajzolódó vonalakból épül fel.

A SET, RESET utasítások paraméterei, vagyis a koordináták (x,y) értelemszerűen csak egész számok lehetnek. A fenti programokban azonban több helyen is a programmal számoltattuk ki a koordinátákat, s a számitás eredménye ritkán egész szám. Az egész számmá alakitás könnyű az INT(a) függvény segitségével, de a gépek automatikusan elvégzik ezt az átalakitást, azaz pl.

45.875 - ből 45 ,

-12.325 - ből -13 lesz, amikor grafikus utasitás paramétereként szerepel. A PRIMO gépek azonban nem tudják értelmezni a negativ koordinátákat, csak a képernyő látható részére szabad cimezni, különben hibajelzést kapunk (Commodore gépeknél megengedett a negatiV érték).

A TVC gépek PLOT utasitása után több koordinátapárt felsorolhatunk. Az összetartozó párok között vessző kell legyen, az egyes koordinátapárokat is vesszővel választva el pontokat rajzol, pontosvesszővel elválasztva viszont egyenessel köti össze a pontokat. gy a képernyőátló:

```
PLOT,0,0;1023;959
```

A képernyő közepére rajzolt 100 egység sugarú kör programja:

```
10 r=100:u=511:v=479
20 for a=0 to 6.28 step 0.02
30 x=u+r*sin(a)
40 y=u+r*cos(a)
50 plot,x,y;
60 next a
```

Lehetőség van többféle szaggatott vonal rajzolására, melyeket a gépkönyv mintái alapján a SET STYLE utasítással állíthatunk be. Zárt alakzatok kifesthetők a PAINT utasítással:

```
70 plot,511,479,paint
```

Ez a programsor a megrajzolt kört befesti. A PRIMO-nál látott bürettát rajzolja meg az alábbi program:

```
10 plot,600,575;600,200,625,575;625,275
20 plot,600,200;550,200,625,175;550,175
30 plot,550,225;550,125,510,225;510,125
40 plot,510,125;530,100,550,125;530,100
50 plot,510,225;550,225
60 plot,505,225;535,255;535,225
70 plot,505,250;525,250;525,225
80 plot,600,570;625,570
90 set ink 2
100 plot,530,200,paint
110 plot,620,300,paint
```

A 90-es sor a kifestés színét állítja be (ld. 8.3. fejezet)

A Commodore 16 és Plus/4 gépeinek BASIC-nyelve több grafikus utasítással segíti a programozót. Könyvünk ugyan nem akarja pótolni a BASIC-tankönyveket, de az oktató-programok szempontjából legfontosabb grafikus utasítások használatának rövid áttekintése nélkülözhetetlen. A nagyfelbontású és a többszínű üzemmódban azonosan használhatók a grafikus utasítások, csak a képernyő vízszintes felbontásának (x-koordináta) különbségére kell ügyelni. E gépek jó tulajdonsága, hogy a képernyő méretén túlmutató koordináták megadása esetén nem jelez hibát, de a rajzból csak a képernyőterületre eső rész fog látszani. Az utasítások első paramétere (a) a nagyfelbontású üzemmódban csak 0 vagy 1 lehet (törlés vagy rajzolás), többszínű üzemmódban 2 és 3 is lehet, amivel a lehetséges színeket állíthatjuk be. A legfontosabb utasítások:

COLOR

A színek beállítására szolgál, parancs módban is (pl. programírás előtt).

COLOR a,b,c : a-val a szintípust (0-4), b-vel a szint (1-16), c-vel a színerőt (0-6) állíthatjuk be. Az első paraméter (a) értékeinek jelentése:

0 : háttér színe
1 : írás színe 1.
2 : írás színe 2.
3 : írás színe 3.
4 : keret színe

Nagyfelbontású üzemmódban csak az első írás-szín jelentkezik, többszínű üzemmódban a további utasítások első paramétere dönti el, hogy a COLOR-ral megadottakból melyik kerül használatra. Érdekes lehetősége ennek az utasításnak, hogy az előzőleg megrajzolt képen is megváltoztatja a színeket egy új COLOR utasítás.

DRAW
Pontok, vonalak rajzolására alkalmas.

DRAW a,x,y : az (x,y) koordinátákkal pontot rajzol,(a) a COLOR utasításban megadott szint jelenti (ha a=0, akkor töröl!)

DRAW a,x,y TO u,v : egyenes vonalat húz (x,y) és (u,v) pontok közé.

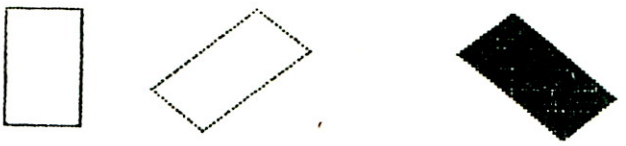
DRAW a TO x,y TO u,v TO ... : a grafikus kurzor aktuális poziciójától (x,y), onnan (u,v), stb. pontig húz egyenest. Az aktuális pozíció mindig a legutólsó rajzolt, vagy törölt pont. A grafikus kurzor pozíciója be is állítható a LOCATE x,y utasítással.

DRAW a,x,y to l;f : (x,y) pontból (l) hosszúságú szakaszt rajzol a függőlegestől jobbra (f) fokkal (l és f között pontosvessző van!). A paramétereket ciklusban változtatva nagyon érdekes rajzokat hozhatunk létre.

BOX
Elforgatható és kifesthető téglalapok rajzolására alkalmas.

BOX a,x,y,u,v,f,s : olyan téglalapot rajzol, melynek bal felső sarka (x,y), jobb alsó sarka (u,v) koordinátájú pontoknál van, és ezt a téglalapot középpontja körül az óramutató járásával megegyező irányba (f) fokkal elforgatja. Ha s=1 akkor be is satírozza, s=0 esetben csak a körvonalakat rajzolja. Az alábbi három programsor eredményét mutatja az ábra:

```
20 box 1,10,10,30,50
30 box 1,60,10,80,50,45
40 box 1,140,10,160,50,135,1
```



CIRCLE
Sokoldalú, bonyolult paraméterezésű utasítás: kört, ellipszist, íveket, sokszögeket rajzol, elforgat.

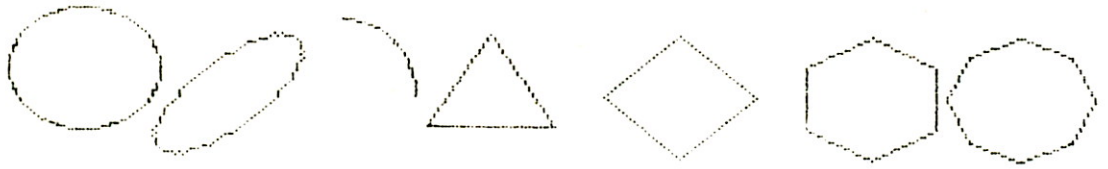
CIRCLE a,u,v,r : (u,v) középpont körül (r) sugarú kört rajzol.

CIRCLE a,u,v,r,p,bf,jf,f,sz: olyan ellipszist rajzol, melynek középpontja (u,v) vízszintes tengelye (r), függőleges tengelye (p) hosszúságú, középpontja körül (f) szöggel van elforgatva. Lehet csak egy ívdarabot rajzolni, ekkor (bf) és (jf) az ív végpontjainak függőlegestől mért bal, illetve jobboldali szögtávolságai. Az (sz) paraméter megadása esetén az ellipszis középpontjától (sz) szögtávolságra lévő pontokat rajzolja meg és egyenesszakasszal köti össze. Az alábbi sorok rendre kört, elforgatott ellipszist, negyed körívet, háromszöget, négyzetet, hatszöget, nyolcszöget rajzolnak.

```
10 circle 1,22,22,20
20 circle 1,60,30,10,25,,,45
30 circle 1,90,30,20,25,0,90
40 circle 1,130,30,20,,,,,120
```

```
50 circle 1,180,30,20,,,,,90
60 circle 1,230,30,20,,,,,60
70 circle 1,270,30,20,,,,,45
```

A programfutás eredménye:



PAINT
Zárt idomok kifestését végző utasítás.

PAINT a,x,y,m : az (x,y) pontot magában foglaló zárt idom belsejét festi ki (a) színtípusnak megfelelően. A festés a vonalig tart, ha m=0 és a vonalat is festi, ha m=1.

A grafikus utasításokban vannak elhagyható paraméterek, ilyenkor helyüket vesszővel kell jelölni, és a gép az úgynevezett alapértelmezéssel számol:

a (színtípus) alapértelmezése: 1 (rajzolás)
f (elforgatás) alapértelmezése: 0 (nincs elforgatás)
s (satírozás) alapértelmezése: 0 (nincs satírozás)
p (ellipszis tengelye) alapértelmezése: r (kör)
bf (ív szögtávolság) alapértelmezése: 0
jf (ív szögtávolság) alapértelmezése: 360 (teljes kör, ellipszis)
sz (szögtávolság) alapértelmezése: 2 (360/2 = 180-szög, azaz kör, ellipszis)

Helykoordináták elhagyásakor az aktuális kurzorpozíciót veszi figyelembe a gép.

A grafikus utasítások mindegyikére látunk példát a továbbiakban. Az alábbi lista összehasonlítás céljából a már látott bürettát rajzolja meg C16-on:

```
10 graphic4,1
70 draw,118,25 to 120,25:draw,125,25 to 127,25
80 draw,118,25 to 120,25:draw,125,25 to 127,25
90 draw,120,25 to 120,95:draw,125,25 to 125,105
100 draw,119,96:draw,124,106
110 draw,119,96 to 110,96:draw,124,106 to110,106
120 draw,110,90 to 110,110:draw,110,110 to 108,115: draw,108,115 to 106,110
130 draw,106,110 to 106,90:draw,106,90 to 110,90
140 draw,109,90 to 109,81 to104,81 to 104,85 to 107,85
150 draw1 to107,90
160 draw,99 ,116:draw,100,116 to 100,150 to 116,150 to 116,116
170 draw,117,116:draw,120,30 to125,30
180 paint 2,122,40,1
190 draw,101,121 to 115,121
200 box 3,101,122,115,149,,1
210 getkey t$:graphic0
```

A Commodore 64-es gép közismert segédprogramja a SIMON'S BASIC, amely mintegy 100 új utasításával nagyszerűen kibővíti a C64 lehetőségeit. Grafikus utasításai hasonlítanak a BASIC V3.5-éhez (fentiek). A program kazettán és lemezen is "terjed", szerző tapasztalatai szerint sajnos hibás másolatok és hibás felhasználói leírások is (1987-ben

végre megjelent egy részletes leírás: Plenge-Szczepanowsky SIMON'S BASIC, DATA BECKER - NOVOTRADE). A betöltött program RUN-nal elindítva rövidesen bejelentkezik, ezután írhatjuk a régi és az új utasításokat is tartalmazó programokat. Természetesen az így írt programok felhasználása előtt mindig be kell tölteni és elindítani a Simon's Basic-et, s csak utána tölteni be a saját programot. E kis kényelmetlenségért bőségesen kárpótol a C16 és Plus/4 grafikáját meghaladó utasításrendszer. Az oktatóprogramok szempontjából legfontosabb utasítások röviden:

COLOUR
A háttér (sh) és a keret (sk) színét állítja be a színkódok értékeivel (0-15):

COLOUR sh,sk

HIRES
Beállítja a nagyfelbontású képernyőt:

HIRES sp,sh : az írás színe (sp) és a háttér színe (sh).

MULTI
A többszínű üzemmódot és színeit állítja be.

MULTI s1,s2,s3 : e három színnel lehet rajzolni.

REC
Téglalap rajzolására alkalmas.

REC x,y,a,b,pl : olyan téglalapot rajzol, amelynek bal felső sarka (x,y), jobb alsó sarka pedig (x+a,y+b) koordinátájú pontoknál van. A (pl) paraméter az úgynevezett plottyp (rajzolási mód), amelynek minden további utasításban az alábbi a jelentése:

Nagyfelbontású üzemmódban:

0 : törlés
1 : rajzolás
2 : invertálás (írás és háttér színét felcseréli)

Többszínű üzemmódban:

0 : törlés
1 : s1 színnel rajzol (MULTI utasításból)
2 : s2 színnel rajzol
3 : s3 színnel rajzol
4 : invertálás (háttérszín inverze s3, s1 inverze s2, s2 inverze s1, s3 inverze a háttérszín)

PLOT
Pontok rajzolására szolgál.

PLOT x,y,pl : az (x,y) koordinátájú pontot (pl) szerint állítja be.

LINE
Egyenesszakaszt rajzol.

LINE x,y,u,v,pl : az (x,y) és (u,v) koordinátájú pontokat köti össze.

CIRCLE
Kör és ellipszis rajzolására alkalmas.

CIRCLE x,y,r,p,pl : (x,y) középpontú, (r) vizszintes, (p) függőleges tengelyű ellipszist rajzol. Kört kapunk, ha r=p

ARC
Ívek és sokszögek rajzolására alkalmas.

ARC x,y,bf,jf,i,r,p,pl : az (x,y) középpontú, (r) és (p) tengelyű ellipszisből rajzol olyan ívet, melyet a függőleges tengelytől balra mért (bf) és jobbra mért (jf) szögek határoznak meg. Az ellipszis úgy áll össze pontokból, hogy először a (bf) szöghöz tartozó pontot rajzolja a gép, majd a (bf+i), (bf+2i), stb. szögekhez tartozókat, majd a pontokat egyenesszakaszokkal köti össze. Minél kisebb (i), annál "szebb" az ellipszis. Nagyobb (i) értékekkel sokszögeket rajzolhatunk (r=p, azaz kör esetén szabályos sokszögeket).

PAINT
Zárt alakzat belsejét festi ki.

PAINT x,y,pl : (x,y) pontból indulva fest a határoló vonalakig.

BLOCK
Kifestett téglalapot rajzol.

BLOCK x,y,u,v,pl : az (x,y) bal felső sarkú és (u,v) jobb alsó sarkú téglalap minden pontjára érvényesíti (pl) hatását.

ROT
A ROT után DRAW utasítással rajzolt alakzatot forgatja és nagyítja.

ROT r,s : az alakzatot r*45 fokkal forgatja el a DRAW utasításbeli kezdőpont körül. (r) lehetséges értékei: 0 - 7. A nagyítás mértéke : (s)-szeres.

DRAW
Érdekes rajzolási mód, a ceruza papíron siklását utánozza.

DRAW <sztring>,x,y,pl : az (x,y) ponttól indulva a sztringben lévő karakterek által jelölt irányba halad és/vagy rajzol. A sztring összesen 74 karaktert tartalmazhat, melyek az alábbiak lehetnek:

0 : kurzor jobbra
1 : kurzor fel
2 : kurzor le
3 : kurzor balra
5 : pont és kurzor jobbra
6 : pont és kurzor fel

7 : pont és kurzor le
8 : pont és kurzor balra
9 : a rajzolás befejezése

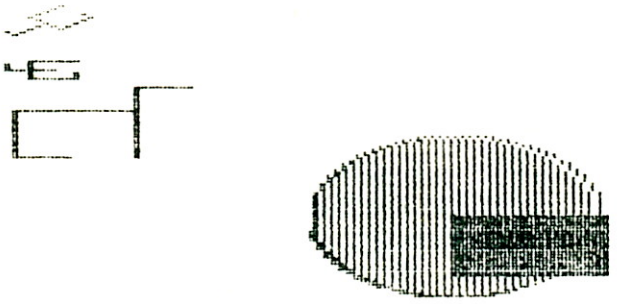
A DRAW utasítás hatása csak egy előzetes ROT utasítás után lesz látható – természetesen a ROT utasításban a forgatás és nagyítás nem kötelező, ROT 0,1 változatlanul hagyja az alakzatot.

Mint láthattuk, a Simon's Basic grafikus utasításai valamivel egyszerűbbek, mint a BASIC V3.5 tömörítettebb és így kevesebb utasítása – de nagyobb lehetőséget biztosítanak. Az alábbi rövid program illusztrálja e lehetőségeket:

```
10 hires 7,0
20 multi 6,9,5
30 line 10,10,100,100,2
40 for a=1 to 500:next
50 hires 7,0
60 line 10,10,100,100,1
70 for a=1 to 500:next
80 multi 0,3,5
90 circle 100,100,50,20,1
100 circle 100,95,50,20,2
110 circle 100,90,50,20,3
120 for a=1 to 500: next
130 hires 7,6:multi 0,3,5
140 arc 100,100,0,90,1,20,28,1
150 arc 100,100,90,150,1,20,28,2
160 arc 100,100,150,190,1,20,28,3
170 arc 100,100,190,260,1,20,28,3
180 arc 100,100,260,360,1,20,28,1
190 block 100,100,120,120,3
200 rot 1,2
210 draw "577886677224356679",50,30,3
215 for a=1 to 500: next
220 rot 2,3
230 draw "577886677224356679",50,50,3
235 for a=1 to 500: next
240 rot 4,8
250 draw "577886677224356679",50,80,3
260 paint 90,90,2
270 goto 270
```

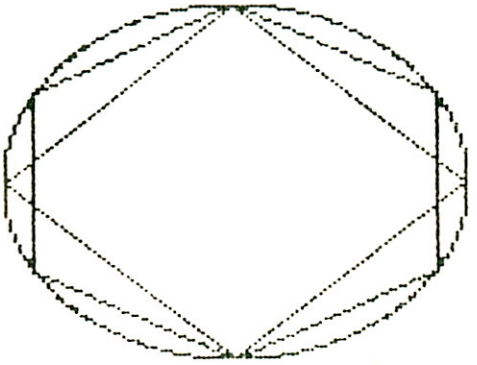
A program 10-es és 20-as sora beállítja a többszínű üzemmódot és színeit, majd az 50-es sor nagyfelbontású üzemmódot állít be. A 30-as és 60-as sor azonos koordinátákkal rajzol egyenest (színük más), ezen összehasonlíthatjuk a kétféle grafikus módot. A 90-110-es sorok három eltérő színű ellipszist rajzolnak. A 140-180-sorok különböző színű ívdarabokból rajzolnak egy ellipszist, majd bele egy kitöltött téglalapot. A 200-250 sorok a DRAW és ROT hatását mutatják, a 260-as sor besatírozza az ellipszis téglalap által szabadon hagyott területét másik színnel. A 140-260 sorok

eredményét mutatja az ábra:



A program 280-as sora újra nagyfelbontású képernyőt állít be, a további sorok egy kört és a bele írható szabályos hatszöget, illetve négyzetet rajzolják meg. Kinyomtatásnál a rajzok arányai torzulnak a nyomtató soremelésének megfelelően:

```
280 hires 0,1
290 arc 150,100,0,360,1,50,50,1
300 arc 150,100,0,360,60,50,50,1
310 arc 150,100,0,360,90,50,50,1
400 goto 400
```



Oktatóprogramok ábráinál fontos a megfelelő szövegek elhelyezése. A PRIMO és TVC gépeknél csak egyféle képernyő van, amelyen rajzok és karakterek is elhelyezhetők az információs modulnál látott kiíratási módokkal. A Commodore gépek grafikus üzemmódjaiban nem ilyen egyszerű a helyzet. A C16 és Plus/4 vágott képernyő esetén az alsó öt sort karakteresen használja, így ott a megszokott módon lehet kommunikálni. Bevihetünk azonban a grafikus képernyőterületre is karaktereket a CHAR utasítással:

CHAR a,x,y,<sztring>,m

(a) jelű színnel (0-3 többszínű, 0-1 nagyfelbontású üzemmódban) (x) oszlop, (y) sorszámmal kezdődően kiírja a <sztring> karaktereit. (m) alapértelmezése: 0, normál írás, ha m=1, akkor inverz írás.

E gépek a CHAR utasítással csak a nagybetű/grafikus karakterkészletet tudják használni, ha kisbetűket és ékezetes betűket, vagy definiált karaktereket szeretnénk a grafikus képernyőre íratni, akkor a megfelelő karakterképek helyét közölni kell a géppel. A tár $02E4 (740) címe tartalmazza a megfelelő mutatót, ami bekapcsolás után $D0 (208), vagyis a karakter-ROM kezdőcímének felső byte-ja. A kisbetűs karakterkészlet $D400 címen kezdődik, ennek felső byte-ja $D4 (212), tehát ezt az értéket kell beírni $02E4 címre. Az átírás monitorban és BASIC-ben is lehetséges. BASIC-ben (a program elején):

```
10 POKE 740,212
```

Amennyiben módosított karakterkészletet helyeztünk el egy 4k-s (2k-s) blokkban, akkor természetesen a megfelelő kezdőcím felső byte-ját kell beírni.

A Simon's Basic változatosabb lehetőséget biztosít a C64-nek feliratozásra két utasítással:

CHAR x,y,k,pl,n

Az (x,y) koordinátájú pontból indulva jobbra és lefelé elhelyezi a (k) karakterkódnak megfelelő karaktert függőlegesen (n)-szeresre nagyítva (n: 1-8).

TEXT x,y,<sztring>,pl,n,i

Az (x,y) koordinátájú pontból indulva jobbra, lefelé elhelyezi a <sztring> karaktersorozatot. Itt (n) a függőleges nagyítás mértéke (1-7), (i-8) a karakterek közötti pontközöket jeleni. Amennyiben i=8, akkor a szokásos módon egymás mellé íródnak a karakterek, ha i=16, akkor egy-egy szóköz kerül közéjük (i=12: fél szóköz, stb.), ha i<8, akkor egymásba íródnak a karakterek. Az alábbi rövid program eredményét mutatja az ábra (tényleges méretben):

```
10 hires 0,1:multi 2,3,4
20 char 100,100,83,3,4
30 text 0,100,"C64",1,7,32
40 goto 40
```

Az eddigiekből látható, hogy a pontgrafika segítségével tetszőleges rajzok, Commodore és TVC gépeken a valóságot közelítő színes képek is létrehozhatók. A programok futtatását összehasonlítva egy "profi" játékprogrammal, az is világos, hogy részletgazdag, finom képekhez a BASIC lassú, ilyeneket csak gépi kódban, esetleg jó grafikai segédprogrammal érdemes tervezni. Oktatási célból azonban a jelenségek lényegét kell ábrázolni, szimulálni - ezért általában elegendő a leegyszerűsitett, kevésbé valósághű rajz. A dolgok természetes mivoltának bemutatása nem a számítógép feladata, erre szolgálnak a diaképek, filmek, videofelvételek - amikor pedig lehetséges, a valódi bemutatás.

8.2.2. Mozgatás

A rajzok mozgatása történhet a karakterek mozgatásához hasonlóan (PRIMO, TVC), de egyes gépek alapvetően más lehetőségeket is kínálnak (pl. a C64 sprite-jai: következő fejezet).

A C16 és Plus/4 utasításrendszerében a GSHAPE, SSHAPE utasításpár segíti a mozgatást, illetve a rajzok ismétlését, többszörözését:

SSHAPE A$,x,y,u,v,

A$ sztringváltozóban tárolja az (x,y) bal felső, (u,v) jobb alsó sarkú téglalap alakú képernyőterület tartalmát. Mivel a sztringáltozók maximális hossza 255 karakter, ezért a tárolható terület nem lehet bármekkora, de a mozgatás sebessége érdekében is a lehető legkisebb terület tárolására kell törekedni. (u,v) koordináták elhagyhatók, ekkor a grafikus kurzor helyzete határozza meg a téglalap jobb alsó sarkát.

GSHAPE A$,x,y,m

Az SSHAPE utasítással A$ sztringáltozóban tárolt téglalap alakú képet másolja vissza, bal felső sarkát (x,y) koordinátájú pontba helyezve. A másolás módját (m) értéke határozza meg. Ha a másolandó terület egy pontjának értéke (a), az új terület megfelelő pontjának értéke (b), akkor a mód szerint az új érték a következő (bináris, tehát a,b értéke: 0 vagy 1):

mód(m)	új érték
0	a (egyszerű másolás)
1	1-a (inverz)
2	a OR b (logikai vagy)
3	a AND b (logikai és)
4	a EOR b (logikai kizáró vagy)

Az (m) elhagyható, alapértelmezése: 0.

A két utasítás egyszerű alkalmazásához elég annyit tudni, hogy m=0 esetén kirajzolja a kívánt helyre a téglalap alakú képet, m=1 esetén inverzét rajzolja, m=4 esetén pedig törli az ott lévő azonos képet.

Itt kell kitérnünk a fenti logikai operátorokra. Az AND és OR egyszerűen és praktikusan használható feltételek összekapcsolására, szó szerint a magyar 'és' 'vagy' hétköznapi jelentésének megfelelően:

IF A=B AND C=D THEN PRINT "JÓ"

A programsor akkor írja ki a "JÓ"-t, ha A=B és C=D (mindkettő igaz).

IF A>10 OR A<0 THEN PRINT "Rossz!"

Ez a sor pedig akkor ad "Rossz!" minősítést, ha az egyik feltétel teljesül, azaz (A) értéke 10-nél nagyobb, vagy 0-nál kisebb.

A mozgatási folyamat a kép kirajzolásának és törlésének váltogatásából tevődik össze, természetesen minden újabb rajzolás, törlés helyzete más és más. A törlésre a SSHAPE utasítás m=4 paraméterezése az egyik lehetőség. Bizonyos esetekben jobb (gyorsabb) lehet egy BOX utasítással rajzolt alapszínű kitöltött téglalap, amely mozgásában szorosan követi az SSHAPE által rajzolt területet. A téglalap legjobb méretének megállapításához először írás színű téglalapot érdemes használni, ekkor pontosan látjuk

a törölt területet. Amikor beállítottuk a megfelelő paramétereket, akkor a BOX utasításban az írás színét háttérszínre cseréljük, így már törölni fog. Gyors (kevésbé villódzó) törlési lehetőséget ad a CHAR utasítás is: szóközöket téve a törlendő helyre. Hátránya, hogy a szóközöket csak karakterpozícióknak megfelelően helyezhetjük el. Az SSHAPE utasítás m=0 esetén természetesen automatikusan törli azt a területet, ahová rajzol. A három törlési módot érdemes kipróbálni, és a próba alapján választani ki az adott feladathoz leginkább alkalmasat. Ilyen program írásakor az elkészült programrészeket (egy-egy rajz) azonnal érdemes kipróbálni, mert ritkán sikerül pontosan eltalálni a szükséges koordinátákat - a kipróbálás alapján lehet módosítgatni. A próbálgatás érdekében magas sorszámmal írjuk be már a program elején:

```
1000 getkey t$:graphic 0:end
```

Ugyanis a program lefutása után megmarad a grafikus üzemmód, tehát nem látjuk amit a billentyűzeten írunk. Amennyiben a fenti sor a program utolsó sora, gombnyomásra visszaáll a karakteres üzemmód, írhatunk, listázhatunk. GRAPHIC 0 utasítás esetén nem vész el a grafikus képernyőtartalom, GRAPHIC 1 hatására visszakapjuk, míg GRAPHIC CLR (GRAPHIC1,1) utasítás töröl is!

A grafikáról eddig elmondottakat néhány rövid, de önállóan is használható, s az olvasó által saját céljára könnyen módosítható programmal illusztráljuk. E programok listáit úgy írtuk, hogy könnyen érthetőek legyenek, kis gyakorlat után fejleszthetők, tömöríthetők.

A "Kristály" című program egy ionkristályt rajzol, amelynek néhány ionját a poláris oldószer (víz) molekulái körbeveszik (hidratálják):

```
10 input"sebesség:";a
20 print chr$(27):print chr$(82):graphic 1,1
30 draw 1,20,20 to24,20:draw 1,22,18 to22,22: circle1,22,20,5,4: sshape a$,17,16,27
,24
40 draw 1,50,20 to 54,20:circle 1,52,20,5,4: sshapeb$,47,16,57,24
50 draw 1,80,20 to 84,20:draw 1,87,20 to90,20: draw1,82,18 to 82,22:circle 1,85,20,8
,4
60 sshape c$,77,16,93,24
70 draw,120,20 to 123,20:draw,126,20 to 130,20: draw,128,18 to 128,22: circle ,125
,20,8,4
80 sshape d$,117,16,133,24
90 draw,160,16 to 160,20:draw,158,22 to 162,22:draw,158,18 to
162,18:circle,160,20,4,6
100 sshape e$,156,14,164,26
110 draw,188,18 to 192,18:draw,190,21 to 190,25: draw,188,23 to
192,23:circle,190,20,4,6
120 sshape f$,186,14,194,26
130 char 1,2,4,"a  b  c  d  e  f "
140 getkey x$:graphic 1,1
150 gshape a$,50,150:gshape b$,150,150:gshape b$,50,80: gshape a$,150,80
160 gshape a$,100,45:gshape b$,200,45:gshape a$,200,115: gshape b$,100,115
170 draw 1,65,155 to 145,155:draw 1,65,85 to 145,85: draw1,115,50 to 195,50
180 draw 1,115,120 to 195,120:draw 1,56,147 to 56,90: draw1,156,147 to 156,90
190 draw 1,206,112 to 206,55:draw 1,106,112 to 106,55
200 draw 1,62,148 to 95 ,125:draw 1,62,78 to 95,55: draw1,162,148 to 195,125
210 draw 1,162,78 to 195,55
220 for x=50to 100 step a
230 box 0,90,0,110,x/3-3,,1
240 gshape e$,100  ,x/3-2
```

```
250 gshape f$,200  ,x/3-2:box 0,190,0,210,x/3-3,,1
260 gshape c$,313-x,45  :box 0,329-x,42,330+a  -x,58,,1
270 gshape c$,x-20 ,45  :box 0,x-20,42,x-21-a  ,58,,1
280 gshape d$,x-70,80  :box 0,x-70,77,x-71-a,93,,1
290 gshape d$,263-x,80   :box 0,279-x,77,280+a-x,93,,1
300 if x>70 then   draw 1,206,112 to 206,55
310 box 0,0,60,x-51,72,,1
320 gshape f$,x-50,60:
330 gshape e$,150,x/3+33  :box 0,140,x/3+25+a,160,x/3+32,,1
340 draw 1,115,50 to 195,50
350 next x
1000 getkey a$:graphic0:end
```

A program első sora kérdez a mozgatás kívánt sebességére (a), amely a mozgató ciklus lépésköze lesz (220-as sor). A 20-as sor a rosszabb TV-készülékekre tekintettel csökkenti a képméretet, parancsban ugyanez két gombnyomás: [ESC] [R].

A 30-130-as sorok kirajzolják a képernyő tetejére a szükséges részecskéket, és SSHAPE utasításokkal sztringekbe tárolják a megfelelő képernyőterületeket.

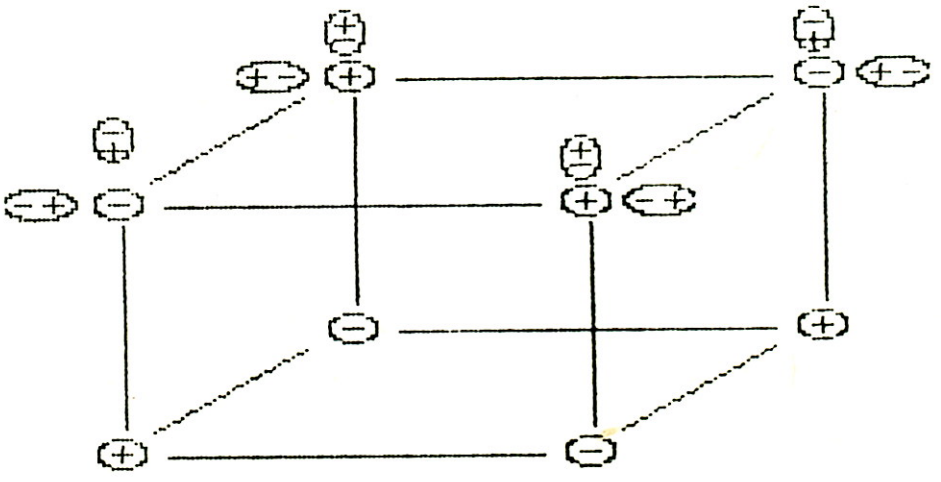
A 140-es sor gombnyomásig vár, majd a grafikus képernyő tartalmát törli, és újra beállítja a grafikus üzemmódot. A már sztringekben tárolt képernyőterületek tartalma természetesen megmarad.

A 150-es és 160-as sor a kristály rácspontjaiban elhelyezi az ionokat (a$, b$), majd a 170-210-es sorok kirajzolják a kristály éleit.

A 220-350-es sorok tartalmazzák a mozgató ciklust. Az e$ és f$ sztringek esetén először töröl (230 és 310), azután rajzolja az új pozícióba az iont (240 és 320), míg a többinél először kirajzolja az újat, azután törli a régit. A két módszer közötti különbség nem feltűnő.

A 300-as es 340-es sorok a mozgó ion által kitörölt egyenesszakaszokat rajzolják újra.

A programfutás egyik fázisának képét mutatja az alábbi ábra:



Az "Ionmozgatás" című program a kurzormozgató billentyűkkel való irányítást mutatja be grafikus képernyőn. A program két részecskét rajzol ki, megkérdezi a mozgatás kívánt sebességét. Mozgatás közben bármikor lehetővé teszi a részecske adott helyen történő rögzítését, és váltást a másik részecskére. Végső soron tetszés szerinti számú részecskével "népesíthetjük" be a képernyőt, létrehozva különböző alakzatokat, modelleket.

```
110 graphic 2,1
120 draw1,3  ,1 to 11,1:draw1,13,3 to 13,5:draw1,11,7 to 3,7:draw1,1,5 to 1,3
130 draw,2,2:draw,12,2:draw,12,6:draw,2,6
140 draw,4,3:draw,4,4:draw,4,5:draw,3,4:draw,6,4: draw,5,4: draw,9,4: draw,10,4:
draw,11,4
150 sshape a$,1,1,13,7
160 circle,220,10,5 ,3
170 draw,220,8 to 220,12:draw,218,10 to 222,10
180 sshape b$,215,7,225,13
190 input"sebesség=";s
200 u=10 :v=10 : gshape a$,u,v
210 getkey i$
220 gshape a$,u,v,4
230 i=asc(i$)
240 if i=84 then 450
250 if i=29 then u=u+s
260 if i=157 then u=u-s
270 if i=17  then v=v+s
280 if i=145 then v=v-s
290 if i=32 then gshape a$,u,v:goto 200
300 if i=83 then graphic0:end
310 if i=66 then 340
320 gshape a$,u,v:goto210
340 u=10:v=10:gshape b$,u,v
350 getkey j$:j=asc(j$):gshape b$,u,v,4
360 if j=84 then 450
370 if j =29 then u=u+s
380 if j =157 then u=u-s
390 if j =17 then v=v+s
400 if j=145 then v=v-s
410 if j=32 then gshape b$,u,v:goto 340
420 if j=65 then 200
430 if j=83 then graphic 0:end
440 gshape b$,u,v   :goto 350
```

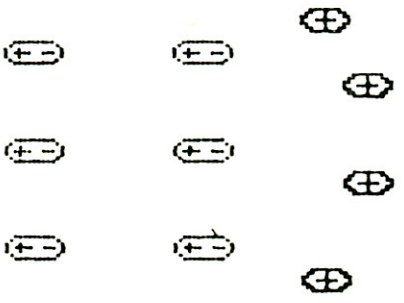
A 120-180 sorok kirajzolnak egy iont és egy poláris molekulát, a rajzokat letárolják két sztringben (a$ és b$).

A 190-es sorban bekért sebesség-adat a koordinátaértékek változtatásának mértéke lesz (250-280 sorok).

A 200-as és a 340-es sor rajzolja ki a mozgatni kívánt részecskét induló helyzetébe, a mozgató-rajzoló ciklusok: 210-320 és 350-440 sorok.

A rajzok törlése az új koordináták beadása után a GSHAPE utasítás m=4 paraméterezésével történik: 220-as és 350-es sorok.

A billentyűlekérdezés a már látott módon történik: a 230-as és 350-es sor a billentyűkód értékét numerikus változóvá alakitja. A négy kurzormozgató billentyű lenyomása esetén a megfelelő koordináta-átirást végzi el, [SPACE] lenyomása esetén (i=32) az előző koordinátáknak megfelelő helyen kirajzolja, és a kiinduló helyzetbe (200-as, illetve 340-es sor) ugorva újra kezdi a mozgatást. [A], vagy [B] betű lenyomása esetén a másik részecskét mozgató ciklus kezdetére lép: 310-es, 420-as sorok, míg [T] betű tovább küldi a programot a következő részre, amely ezen a listán nem szerepel: 240-es, 360-as sorok. Az alábbi ábra egy kialakitható képet mutat:



A látott mozgatási lehetőségek úgy is kombinálhatók, hogy egyszerre több mozgó alakzat legyen a képernyőn. Programból mozgatni elvileg bármennyi alakzatot lehet, de számukkal arányosan csökken a mozgás sebessége; az ésszerűség szab határt. Kurzormozgató billentyűrendszer egy van, joystick csatlakozás kettő, vagyis e módszerekkel egy, illetve két független mozgás érhető el. Láttuk, hogy bár egyszerre csak egy ciklus lehet érvényben, a ciklusváltozóból matematikai műveletekkel több irányú mozgást is létre lehet hozni. Felhasználó által vezérelt mozgásnál nem szabad ezt az eljárást alkalmazni, mert nehéz követni az egy gombnyomásra másfelé mozgó különböző alakzatokat. Egy pszichológiai vizsgálat (pl. figyelemmegosztás) céljára azonban készithetünk ilyen programot is.

Több alakzat mozgatásánál természetesen vetődik fel az ütközés kérdése. Amennyiben azonos képernyőterületre több alakzat 'érkezik', mindig az utóbb érkező törli az előtte ott lévőt karaktermozgatás, illetve GSHAPE utasitásban m=0 esetén. A GSHAPE (m) paraméterének más értékei az egyes képpontok bitjei között az az elözőekben leirt logikai műveleteket eredményezik, legjobb kipróbálni a hatásukat.

Két alakzat ütközésének vizsgálata a megfelelő koordináták értékei közötti különbségre alapozható. Ismerve az SSHAPE utasitással elmentett területet, kiszámitható, hogy két alakzat mekkora koordináta-különbség esetén érinti meg egymást: a koordináták a terület bal felső pontját adják meg. Természetesen itt is a kipróbálás dönt végül.

Az alábbi program az ütközés legegyszerűbb esetét mutatja be: két kör halad egymással szemben, összeütközve "felrobbannak":

```
2000 rem"Ütközés"
2010 graphic2,1
2012 circle 1,5,5,3
2014 circle 1,25,5,4
2020 sshape a$,0,0,10,10
2030 sshape b$,20,0,30,10
2050 rem"mozgatás"
2060 for z=1 to 300 step 5 :y=50
2065 x=z-5:v=305-z
```

```
2070 gshape a$,x,y,4
2080 gshape a$,x+5,y
2090 gshape b$,v,y,4
2100 gshape b$,v-5,y
2120 if abs(x-v)<20 then 2300
2200 next z
2300 graphic 0:print"FELROBBANTAM!"
```

A programban csak a 2120-as sor jelent újdonságot: azt a közelséget adja meg, amelynél már 'robban'. Itt nem engedi meg a program a teljes érintkezést. Ha az alakzatok pontos összeérését kívánjuk észlelni, akkor tekintettel kell lenni a következőkre:

- a mozgó területek nagysága: 10x10
- a ciklusváltozó lépésköze legyen 1
- a koordináták különbsége érintkezéskor: 10 (vízszintesen)

Ennek érdekében módosítsuk az alábbi két sort:

```
2060 for z= 1 to 300: y=50
2120 if abs(x-v)<11 then 2300
```

A "Kergetés" című program egy, a fentihez hasonlóan mozgó kör másik kör általi üldözését teszi lehetővé, az üldöző kurzormozgatókkal, vagy joystickkel irányítható:

```
3000 rem"kergetés"
3002 print"<CLR>":print"Mivel játszol?"
3004 print:print"1: kurzormozgató billentyűk"
3005 print"2: botkormány (joystick)"
3006 getkey v$
3007 if v$="1"then 3010
3008 if v$="2"then 3500
3009 print"1,2-vel válaszolj!":goto 3006
3010 graphic2,1
3012 circle 1,5,5,3
3014 circle 1,25,5,4
3020 sshape a$,0,0,10,10
3030 sshape b$,20,0,30,10
3040 u=100:v=100:s=10
3060 for z=1 to 300 step 5 :y=50
3065 x=z-5:
3070 gshape a$,x,y,4
3080 gshape a$,x+5,y
3120 getkey m$
3130 i=asc(m$)
3140 gshape b$,u,v,4
3150 if i=29  then u=u+s
3160 if i=157 then u=u-s
3170 if i=17  then v=v+s
3180 if i=145 then v=v-s
3190 gshape b$,u,v
3200 if abs(x-u)<10 and abs(y-v)<10 then 3250
3220 next z
3240 graphic0:print"nem talált":end
3250 graphic0:print"talált!":end
3500 graphic2,1
```

```
3504 u=100:v=100:s=10
3512 circle 1,5,5,3
3514 circle 1,25,5,4
3520 sshape a$,0,0,10,10
3530 sshape b$,20,0,30,10
3532 for z=1 to 300 step 5
3535 x=z-5:y=50
3536 gshape a$,x,y,4
3538 gshape a$,x+5,y
3545 gshape b$,u,v,4
3550 if joy(1)=3   then u=u+s
3560 if joy(1)=7  then u=u-s
3570 if joy(1)=5 then v=v+s
3580 if joy(1)=1 then v=v-s
3582 if joy(1)=4 then u=u+s:v=v+s
3584 if joy(1)=2 then u=u+s:v=v-s
3586 if joy(1)=8 then u=u-s:v=v-s
3588 if joy(1)=6 then u=u-s:v=v+s
3590 gshape b$,u,v
3600 if abs(x-u)<10 and abs(y-v)<10 then 3250
3620 next z
```

Az ilyen programnál különösen kell ügyelni a ciklusszervezésre. A program által mozgatott részecske mozgató ciklusának (3060–3220) magában kell foglalnia a billentyű-lekérdező és koordináta-frissitő ciklust is (3120–3190).

A 3200-as sor a ciklus minden futásánál megvizsgálja az ütközést, amely most csak akkor eredményes, ha a részecskék egymásbahatolnak (részben fedik egymást).

A 3040-es sor meghatározza az irányitható részecske sebességét, de ez a szokásos módon a felhasználó által is meghatározható:

```
3040 u=100:v=100:input"sebesség:";s
```

Ebben az esetben azonban gondoskodni kell megfelelő sebességhatárokról, mert túl nagy lépésköznél esetleg a kergető részecske átugorja az üldözöttet!

A 3240-es sor abban az esetben értékel, ha az üldözött épségben eléri a ciklus végét.

3500-as sorban kezdődik a program másik része, amely a joystickkel való vezérlést választva indul. Innen inditva a programot, teljes programként használhatjuk, a 3500–3530-as sorok elvégzik a rajzolást, sztringgé alakitást úgy, mint a 3010–3030-as sorok.

A programok fejlesztése során törekedni kell a tömöritésre, gyorsitásra. A fenti programban a továbbfejlesztés első lépése lehet – mint gyakorló feladat – az IF...THEN sorok átirása ON...GOTO-ra.

A C16 és Plus/4 gépek grafikus üzemmódba kapcsolásánál gyakran felvillan az előző grafikus képernyő tartalma, illetve szines foltok. E szépséghibát el lehet kerülni a képernyő rövid időre történő elsötétitésével (kikapcsolásával), amely egyébként a mikroprocesszor működését is gyorsitja (pl. kazettáról való betöltésnél). A művelet igen egyszerű:

- elsötétités: POKE 65286,PEEK(65286) AND 239
- visszakapcsolás: POKE 65286,PEEK(65286) OR 16

Pontosan: a TED $FF06 című regiszterének 4. bitjét kell alacsonyra (0), illetve magasra (1) állítani, mert ez vezérli a video chip működését. A fenti megoldással elérjük, hogy csak ennek a bitnek az értéke változzon.

Az elsötétítést érdemes alkalmazni rajzolás közben, ha jobbnak ítéljük a kész rajz pillanatszerű megjelenését. Egy időzítő ciklussal, vagy a belső óra felhasználásával megoldható a rajzok meghatározott ideig tartó megmutatása, majd kérdezés a rajz tartalmáról,s a válasz után a rajz újra bemutatása (vizuális memóriát fejlesztő programok, vetélkedők).

A PRIMO gépek rajzainak mozgatása elvileg hasolóan oldható meg: kirajzolás és törlés állandó váltakozásával. A SET és RESET utasitásokat kell megfelelő ciklusba foglalni, törlésre használható a teljes képet törlő CLR is - ha csak a mozgó ábra van a képernyőn. Az alábbi program egy kis négyzetet mozgat függőlegesen felfelé:

```
10 cls
20 input"sebesség:";b
30 for a=1 to 100 step b
40 set(20,a):set(19,a):set(18,a)
50 set(20,a+1):set(20,a+2):set(20,a+3)
60 set(19,a+3):set(18,a+3)
70 set(18,a+2):set(18,a+1)
80 reset(20,a):reset(19,a):reset(18,a)
90 reset(20,a+1):reset(20,a+2):reset(20,a+3)
100 reset(19,a+3):reset(18,a+3)
110 reset(18,a+2):reset(18,a+3)
120 next a
```

A 40-70-es sorok kirajzolják, majd a 80-110-es sorok törlik a négyzetet. A 70-es sor után érdemes különböző lassító ciklusokat kipróbálva megkeresni a szemnek legkellemesebb mozgást - enélkül igen gyors.

A PRIMO gépek nem rendelkeznek kurzormozgató billentyűkkel, de bármely négy billentyűt használhatjuk a mozgó ábra irányitására. Az előzőekben látott C16-programokhoz hasonló ciklusban kell lekérdezni a billentyűzetet és az eredmény alapján módositani a koordinátákat. Egy másik lekérdezési lehetőség:

```
100 a$=inkey$:if a$="" then 100
110 reset(x,y)
110 if a$="u" then y=y+1
120 if a$="m" then y=y-1
130 if a$="k" then x=x+1
140 if a$="j" then x=x-1
150 set(x,y)
160 goto 100
```

Ez a kis program egy pontot mozgat, kurzormozgató billentyűknek az 'u,m,k,j' betűket választottuk megfelelő elhelyezkedésük miatt. Természetesen nincs akadálya a nyolc irányba való mozgatásnak sem (mint joystickkel) nyolc betű felhasználásával.

Ugyanezt a feladatot TVC gépen egyszerűbben lehet megoldani karakterszerkesztéssel kialakitott négyzettel. Az alábbi program a beépitett joystick segitségével mozgatja a kis négyzetet:

```
10 rem"négyzet mozgatás"
20 cls:graphics 4
30 set character 130,0,255,129,129,129,129,129,129,255,0
40 u=2:v=2:goto 140
50 a$=inkey$:if a$="" then 50
60 if a$=chr$(5) then u=u-1:goto 100
70 if a$=chr$(24) then u=u+1:goto 100
80 if a$=chr$(4) then v=v+1:goto 100
90 if a$=chr$(19) then v=v-1:goto 100
100 if u>22 then u=22
110 if u<1 then u=1
120 if v>30 then v=30
130 if v<1 then v=1
140 print at u,v:" ";chr$(130);" "
150 print at u-1,v:"   "
160 print at u+1,v:"   "
170 goto 520
```

A fejezet példáiból látható, hogy rajzok, alakzatok mozgatásával sok jelenséget lehet grafikusan szimulálni. Zárásképpen néhány ajánlás:

- C16 és Plus/4 gépen az SSHAPE-GSHAPE utasításpárral kisebb, egyszerűbb alakzatok elfogadhatóan mozgathatók akár a program, akár a felhasználó által vezérelve.
- Könnyen megoldható a mozgatott alakzat többszörözése, rögzítése, így változatos, önálló feladatokat adhat a program a tanulóknak.
- Két vagy több alakzat egymáshoz viszonyított helyzetét folyamatosan vizsgálhatja a program (közelség, ütközés), ezzel fizikai, kémiai jelenségek, sportjátékok, stb szimulációja válik lehetővé.
- PRIMO és TVC gépeknél a program kell pótolja a GSHAPE utasítást (illetve TVC-nél a karakterszerkesztés is): minden törlés után újra rajzolni a teljes alakzatot.
- Ilyen programokat úgy érdemes megirni, hogy a felhasználó választhasson a kurzormozgató (esetleg egyéb) billentyűk és a joystick között.

8.2.3. Sprite-grafika

A sprite-ok (=szellemek) a C64 egyedi grafikus lehetőségei, melyek lehetővé teszik BASIC-ben is magas színvonalú mozgásos, grafikus program írását. A C16 és plus/4 GSHAPE-SSHAPE utasításpárja ezt utánozza, de e gépeknél elfogadható minőségben csak gépi kódú programmal lehet utolérni a C64 sprite-jait.

A sprite a képernyő bármely részén megjeleníthető 24x21 képpontból álló alakzatot jelent. A C64 egyszerre 8 független és különböző sprite-ot tud kezelni, melyek fontosabb tulajdonságai:

- színeik különbözhetnek,
- nagyíthatók,
- tetszés szerint mozgathatók,
- egymás közti és háttérrel való takarásuk választható,
- egymással, vagy háttérrel való ütközésük regisztrálható,
- többszínű üzemmód is lehetséges (egy sprite-ban 4 szín).

A sprite-okat a C64 VIC-II chipje (Video Interface Chip) kezeli, amelynek fontosabb memóriacímei (sprite-regiszterek):

Báziscím: 53248 (ehhez kell hozzáadni a regiszterek számait)

regiszter	tartalom
0	0. sprite x-koordinátája
1	0. sprite y-koordinátája
2	1. sprite x-koordinátája
3	1. sprite y-koordinátája
15	7. sprite y-koordinátája
16	sprite-ok MSB-regisztere
21	sprite-ok engedélyezése
23	y-irányú nagyítás
27	sprite-háttér prioritás
28	többszínű üzemmód bekapcsolás
29	x-irányú nagyítás
30	sprite-sprite-ütközés
31	sprite-háttér-ütközés
37	többszínű üzemmód 0.szín
38	többszínű üzemmód 1.szín
39	0. sprite színe
40	1. sprite színe
46	7. sprite színe

Egy sprite 24x21=504 képpontból áll, tehát 504/8=63 byte szükséges alakjának leírásához. A sprite-ok adatait a video-chiphez tartozó 16 k memóriaterületen lehet elhelyezni, amely 64 byte-onként lapokra (blokkokra) van osztva. Az egyes sprite-ok adatainak helyét megadó mutatókat a 2040-2047 címeken kell elhelyezni:

```
2040   2041   2042   2043   2044   2045   2046   2047
 0.     1.     2.     3.     4.     5.     6.     7. sprite lapszáma
```

Az egyes lapok (blokkok) kezdőcímeit a lapszám 64-szerese adja, pl. a 13. lap kezdőcíme: 13x64=832.

A továbbiakban egy későbbi játékprogramban szereplő sprite megszerkesztésén keresztül illusztráljuk az elmondottakat.

Engedélyezzük a 7. sprite-ot, egyúttal megadva a báziscímet is:

```
5 V=53248
10 POKE V+21,128
```

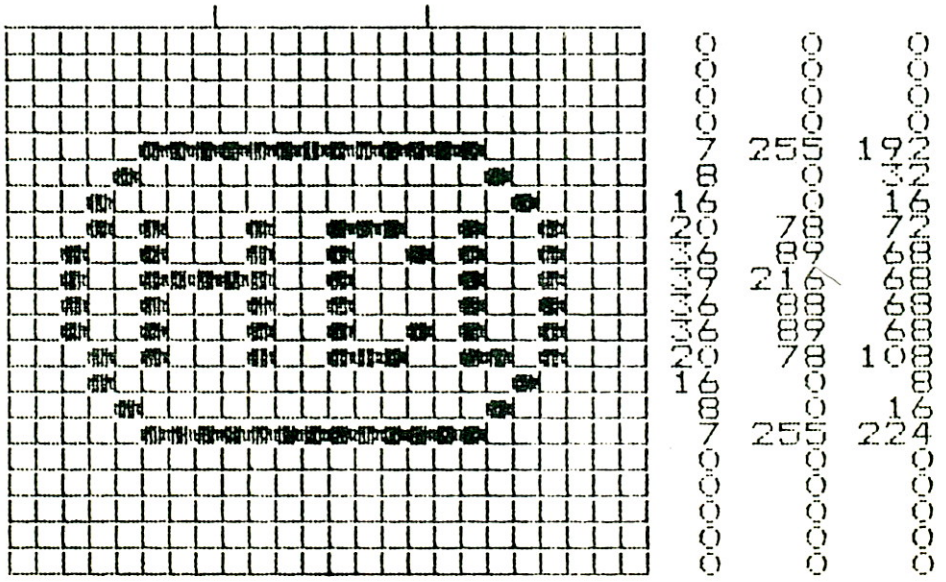
A 21-es regiszter végzi mindegyik sprite engedélyezését, az engedélyezett sprite bitje magas (1), a nem engedélyezett sprite bitje alacsony (0):

7.	6.	5.	4.	3.	2.	1.	0. sprite
1	0	0	0	0	0	0	0
128	64	32	16	8	4	2	1

Mivel csak a 7. sprite-ot engedélyezzük, a byte értéke: 128
Helyezzük el engedélyezett sprite-unk leirását a 13. blokkba:

```
20 POKE 2047,13
```

A 13. blokk (lap) kezdőcime: 13x64=832, tehát a 832-vel kezdődő 63 byte tartalmazza a sprite adatait. A kép megrajzolását 24x21 pontként négyzetrácsos papiron célszerű elvégezni:



A pontminta mellett a soronkénti bitekből összeadott három byte értékeit tüntettük fel, melyeket ilyen hármas csoportokban kell elhelyezni a 13. blokkban. 63 byte-ot egyenként POKE utasitásokkal beirni nagyon fáradságos, a READ-DATA utasitáspár jelentősen megkönnyiti a munkát:

```
30 FOR N=0 TO 62
40 READ A: POKE 832+N,A
50 NEXT N
200 DATA 0,0,0,0,0,0,0,0,0,0,0,0,7,255,192
210 DATA 8,0,32,16,0,16,20,78,72,36,89,68,39,216,68
220 DATA 36,88,68,36,89,68,20,78,108,16,0,8,8,0,16
230 DATA 7,255,224,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

A DATA-sorokat a program végére szokás írni, de bárhol lehet, a READ 'megkeresi'. Amennyiben több sprite-ot használunk, folyamatosan lehet adataikat közölni a DATA-sorokban, csak a 30-50 sorokat kell minden sprite-hoz megismételni a POKE utáni megfelelő címmel. Lehetséges több egyforma sprite-ot készíteni, ekkor a DATA-sorokat nem kell ismételni, az utolsó után elég egy sor:

```
240 RESTORE
```

Ez az utasítás lehetővé teszi, hogy a READ többször kiolvassa ugyanazt az adatsort - és több sprite-ot rajzoljon belőle. Ne felejtsük el, hogy a több sprite-ot előbb engedélyezni kell a 21-es regiszter megfelelő értékével (10-es sor).

A sprite-ok mozgatása a már látott mozgatásokhoz hasonló: koordinátáikat kell ciklusváltozóval, kurzormozgató billentyűkkel, vagy joystickkel átírni, és a 0-15. regiszterekbe beírni POKE-utasításokkal az aktuális koordinátákat:

```
60 FOR X=10 TO 150
70 POKE V+14,X: POKE V+15,X/2
80 NEXT X
```

E három sor a bal felső sarok irányából a képernyő közepe felé mozgatja a sprite-ot.

A képernyő és a sprite-tér (amiben a sprit-ok mozoghatnak) nem esik egybe, utóbbi nagyobb, így a sprite-ok simán úszhatnak be a képbe. A sprite-tér nagysága: 512x256 képpont, ebből látható: 320x200. A sprite-térben úgy helyezkedik el a látható rész, hogy a kétszeresre nagyított sprite is csaknem elfér bárhol a látható területen kívül. A sprite-ok X-koordinátája 0-511, az Y-koordinátája 0-255 között lehet. Mivel egy byte értéke legfeljebb 255 lehet, látható, hogy a X-koordináták teljes tartománya nem adható meg egy byte-on. Ezért első közelítésben az X-koordináta értékét csak 255-ig növelhetjük - a kimaradó terület alkalmas lehet szövegek, adatok közlésére (képernyő jobboldali harmada). Amennyiben a teljes területet szeretnénk használni, vegyük figyelembe a következőket:

- a 16-os regiszter tárolja mindegyik sprite X-koordinátájának legmagasabb helyiértékű bitjét (Most Significant Bit =MSB) az alábbi módon: 0. bit-> 0. sprite, 1.bit-> 1, sprite, stb.
- ha a kérdéses bit magas (1), akkor a 0-16 regiszterben tárolt megfelelő byte-hoz 256-ot adva kapjuk a kérdéses koordinátát.

Egy példa:

```
90 POKE V+16,128
100 FOR X= 0 TO 100
110 POKE V+14,X
120 NEXT X
```

Így a 7. sprite a kép jobb széle felé mozog az X=255 pozíciótól.

Az egyes sprite-ok színe különbözhet (16 szín), de egy sprite kivilágított pontjai azonos színűek alapmódban. A színek beállítására szolgálnak a 39-46. regiszterek, melyekbe a megfelelő színkódokat kell beírni, célszerűen a koordináták beírása előtt:

```
51 POKE V+46,7
```

Ezzel sprite-unk sárga színben jelenik meg.

A többszínű üzemmód négy szint enged meg egy sprite-on belül a vízszintes felbontás csökkenése árán. Ebben az üzemmódban a soronkénti 24 képpont csak párosával használható, vagyis lényegében 12 független dupla szélességű pontból áll egy sor. A színeket a megfelelő regiszterekben tárolt színek közül a pontpárok bit-kombinációival választhatjuk ki:

pontpár (bit-pár)	szín
00	háttérszín
01	többszín 1 (37. regiszter)
11	többszín 2 (38. regiszter)
10	sprite szin (40-46. regiszterek)

A többszínű üzemmódot sprite-onként külön-külön engedélyezhetjük a 28-as regiszter megfelelő bitjeivel: 0=standard, 1=többszínű. A 7. sprite többszínűvé alakitásának első lépése tehát:

```
52 POKE V+28,128
54 POKE V+37,6
56 POKE V+38,2
```

Így sárga (51-es sor), kék (54), piros (56) és háttérszínű pontokból épithetjük fel a sprite-ot. Természetesen a DATA-sorokat is újra kell irnunk. A szerkesztéshez használt négyzetrácsot dupla széles pontokkal kell elkésziteni, amelyben a pontokat bejelöljük a kivánt színekkel, majd beleirjuk a megfelelő bit-párokat:

00	háttér (átlátszó)
01	piros
11	kék
10	sárga

Az alábbi ábra a színek kezdőbetűivel mutatja a négyzetrácsot:

```
H H H H H H H H H H H H
H H H H H H H H H H H H
H H H H H H H H H H H H
H H H H H H H H H H H H
H H H P P P P P P H H H
H H P S S S S S S P H H
H P S S S S S S S S P H
P S S S S S S S S S S P
P S K S K S K K S K S P
P S K S K S K S S K S P
P S K S K S K S S K S P
P S K K K S K S S K S P
P S K S K S K S S K S P
P S K S K S K K S K K P
H P S S S S S S S S S P
H H P S S S S S S S P H
H H H P S S S S S P H H
H H H H H H H H H H H H
H H H H H H H H H H H H
H H H H H H H H H H H H
H H H H H H H H H H H H
```

A munka következő része a betűknek megfelelő bitpárok beírása a négyzetrácsba. Legegyszerűbb átttetsző papirt helyezni a fenti ábrára, és a betűk helyére írni a bitpárokat. Az alábbi ábra már a DATA-sorokba írandó byte-ok értékeit is tartalmazza:

```
000000000000000000000000      0     0     0
000000000000000000000000      0     0     0
000000000000000000000000      0     0     0
000000000000000000000000      0     0     0
000000010101010101000000      1    85    64
000001101010101010010000      6   170   144
000110101010101010100100     26   170   164
011010101010101010101001    106   170   164
011011101110111111011001    110   239   185
011011101110111010111001    110   238   185
011011101110111010111001    110   238   185
011011101110111010111001    110   238   185
011011111111011101011100    111   238   185
011011101110111010111001    110   238   185
011011101110111111011001    110   239   185
000110101010101010100100     26   170   164
000001101010101010010000      6   170   144
000000010101010101000000      1    85    64
000000000000000000000000      0     0     0
000000000000000000000000      0     0     0
000000000000000000000000      0     0     0
000000000000000000000000      0     0     0
```

A sprite-okat egyik, vagy mindkét irányban kétszeresére lehet növelni a 23. és a 29. regiszter beállításával. Amelyik sprite-nak megfelelő bitet magasra (1) állítjuk, a sprite abban az irányban kétszeresére nő. Az alábbi programsor mindkét irányban megnöveli sprite-unkat:

```
58 POKE V+23,128:POKE V+29,128
```

Talán sok volt már a sprite-grafikából, pihenésül írjuk be és futtassuk le az eddig összeállított programot:

```
5 v=53248
10 poke v+21,128
20 poke 2047,13
30 for n=0 to 62
40 read a: poke 832+n,a
50 nextn
51 poke v+46,7
52 poke v+28,128
54 poke v+37,2
56 poke v+38,6
58 poke v+23,128:poke v+29,128
60 for x=10 to 150
70 poke v+14,x: poke v+15, x/2
80 next x
200 data 0,0,0,0,0,0,0,0,0,0,0,0,1,85,64
210 data 6,170,144,26,170,164,106,170,169,110,239,185
220 data 110,238,185,110,238,185,111,238,185,110,238,185
230 data 110,239,185,26,170,164,6,170,144,1,85,64
240 data 0,0,0,0,0,0,0,0,0,0,0,0
```

Mivel egyszerre több sprite mozoghat és egyéb rajzok, karakterek (háttér) is lehetnek a képernyőn, ezért meg kell határozni a prioritásokat, azaz, hogy fedésnél melyik melyiket takarja. A sprite-háttér közti prioritást a 27. regiszter bitjei határozzák meg: 0 esetén a sprite látszik, 1 esetén a háttér. A sprite-ok közötti prioritás a sorszámnak megfelelő: a kisebb sorszámú takarja a nagyobb sorszámút. Ha eddig szerkesztett sprite-unkat a háttér mögött akarjuk mozgatni, írjuk be:

```
15 POKE V+27,128
```

E sor nélkül a háttér előtt mozog minden sprite, hiszen beavatkozás nélkül a regiszterek tartalma: 0.

A VIC-II chip nagyszerű szolgáltatása a sprite-ok közötti ütközés jelzése. Két regiszter megfelelő bitjei jelzik, hogy történt-e ütközés (1), vagy sem (0). Ütközés szempontjából a sprite-nak csak a nem átlátszó (nem háttérszínű) része számít, vagyis csak a megrajzolt alak, nem pedig a 24x21-es pontrács.

A 30-as regiszter annyiadik bitje lesz magas (1), amelyik sorszámú sprite háttérrel ütközött. A 31-es regiszternek pedig az a bitje, ahányadik sprite ütközött bármelyik másik sprite-vel.

PEEK(V+30) =1 , ha a 0. sprite ütközött a háttérrel,
PEEK(V+30) =128, ha a 7. sprite ütközött a háttérrel,
PEEK(V+30)=129, ha a 7. és a 0. sprite is ütközött a háttérrel, stb.

Az alábbi program az előző kibővítése úgy, hogy két sprite mozog és ütközik:

```
5 v=53248 :print"<CLR>":poke v+30,0
10 poke v+21,129
20 poke 2047,13:poke 2040,13
30 for n=0 to 62
40 read a: poke 832+n,a
50 nextn
51 poke v+46,7
52 poke v+28,129
54 poke v+37,2
56 poke v+38,6
58 poke v+23,128:poke v+29,128
60 for x=10 to 150
70 poke v+14,x: poke v+15, x/2+50
75 poke v,250-x:poke v+1,x/2+50
78 if peek(v+30) =129 then 100
80 next x
100 print"durr!":poke v,0:pokev+1,150:poke v+14,0:poke v+15,0
200 data 0,0,0,0,0,0,0,0,0,0,0,0,1,85,64
210 data 6,170,144,26,170,164,106,170,169,110,239,185
220 data 110,238,185,110,238,185,111,238,185,110,238,185
230 data 110,239,185,26,170,164,6,170,144,1,85,64
240 data 0,0,0,0,0,0,0,0,0,0,0,0
250 restore
```

Az 5-ös sorban töröljük a képernyőt és nullázzuk az ütközést jelző regisztert. A nullázást minden ütközés vizsgálat előtt meg kell tenni.

A 10-es sorban a 129 (128+1) engedélyezi a 0. és a 7. sprite-ot.

A 20-as sort úgy módosítottuk, hogy a 2040-es címen is elhelyeztünk egy sprite-mutatót, így a 0. sprite ugyanúgy épül fel, mint a 7. (13-as blokk).

Az 58-as sort, amelyik a nagyítást engedélyezte, nem módosítjuk, így új sprite-unk 24x21 pont méretű marad.

A 75. sor a 0. sprite mozgatását végzi a 7. sprite útját keresztező irányba.

A 78-as sor minden ciklusban megvizsgálja az ütközési regiszter tartalmát, s amennyiben a 0. és a 7. sprite bitje magas, akkor a 100. sorra lép.

A 100-as sor kiírja: DURR! és a két sprite-ot áthelyezi nem látható területre.

A 250-es sor a READ számára engedélyezi az egyszer már kiolvasott adatok újraolvasását a 0. sprite felépítése céljára. Mivel itt a két sprite a program által vezérelt pályán mozog, biztos az ütközés, ezért nem kell a program elágaztatásáról gondoskodni ellenkező esetre.

Felmerülhet, miért kell a 100-as sorban új helyre küldeni a két sprite-ot? Mivel a program megállása nem törli a sprite-okat, ezért a következő program-indításnál az ütközés helyén felvillannának a sprite-ok, újabb ütközést jelezve. Próbáljuk meg e négy POKE utasítás nélkül is a programot!

A népszerű Simon's Basic segédprogram természetesen a sprite-grafikát is támogatja. Előnye főleg a mozgatás sebességében mutatkozik: sokkal gyorsabb a BASIC-programnál. A sprite adatainak megadásánál mentesít a fáradságos számolástól - más vonatkozásban nem egyszerűbb az előzőeknél. A Simon's Basic legfontosabb sprite-utasításai:

DESIGN
Sprite definiálását biztosítja.

DESIGN a,c : standard (a=0), vagy többszínű sprite-nak (a=1) foglal helyet a memóriában, (c) címtől kezdve, amely 64 többszöröse lehet, úgy, mint BASIC-ben.
A DESIGN után huszonegy soron át @ jellel kezdve a sorokat írhatjuk le a sprite alakját, ha a=0, akkor 24, ha a=1, akkor 12 karakterrel soronként. Standard üzemmódban pont (.) jelzi a háttérszínű, B betű a szín-regiszterben megadott színt. Többszínű sprite esetén még a C és D betűket használhatjuk a CMOB utasításban megadott két szín jelölésére.

CMOB
A többszínű sprite-ok további színeit adja meg.

CMOB c,d : A DESIGN után C és D betűkkel megadott pontok veszik fel a (c), illetve (d) színeket (értékük 0-15 között lehet).

MOB SET
Létrehoz egy sprite-ot.

MOB SET sz,lsz,sz,pr,m : az (sz) sorszámú sprite-ot az (lsz) lapszámú memóriaterületről (sz) színnel engedélyezi felépíteni. A háttérrel szembeni prioritás (pr) lehet 0 (sprite látszik), vagy 1 (háttér látszik), az üzemmód (m) lehet 0 (standard), vagy 1 (többszínű).

MMOB

Képernyőre állít és mozgat egy sprite-ot.

MMOB sz,x,y,u,v,n,seb : az (sz) sorszámú sprite-ot az (x,y) pontból (u,v) pontba mozgatja, (seb) sebességgel, és (n) módon nagyítva. A koordináták határértékei: 0-511(X) és 0-255(Y). A sebesség értéke 1-255 között lehet, de ebből 255 a leglassúbb! Az (n) paraméter lehetséges értékei:

0: normál méret (24x21)
1: X-irányú kétszeres nagyítás (48x21)
2: Y-irányú kétszeres nagyítás (24x42)
3: mindkét irányban kétszeres nagyítás (48x42)

RLOCMOB
Képernyőn lévő sprite mozgatását végzi

RLOCMOB sz,u,v,n,seb : az (sz) sorszámú sprite-ot aktuális helyétől (u,v) pontba mozgatja, paraméterei a MMOB-nál leírtakkal megegyeznek.

MOB OFF
Sprite kikapcsolását végzi.

MOB OFF sz : (sz) sorszámú sprite eltűnik.

DETECT
Előkészíti az ütközés lekérdezését.

DETECT ü : ü=1 esetén a sprite-háttér, ü=0 esetén a sprite-sprite ütközés regisztert törli.

CHECK
A függvény értékéből az ütközésre lehet következtetni.

CHECK(sz1,sz2) : e függvény értéke 0, ha az 1-es és a 2-es sprite ütközött egymással, vagy egyszerre másik sprite-al (utóbbinak kicsi az esélye).
CHECK(0) : a függvény értéke 0, ha valamelyik sprite a háttérrel ütközött.

A Simon's Basic előnyei a sprite adatainak bevitelében és a mozgatás gyorsaságában rejlenek - mint említettük. Az alábbi program e két előnyt illusztrálja: elkészítése tized-annyi ideig sem tartott, mint a 'HCI'-programé, mert nem kell a byte-okat számolni. A mozgási sebességet érzékelendő, a 370-es sor utolsó paraméterét cseréljük 200-ról 1-re...

A program egyébként egy piros kétpettyes katicabogarat másztat a képernyő aljáról a közepére. Természetesen lehet utánozni a rajzfilmet finomabban is: a katicabogár lábai mozoghatnak menet közben! Az ilyen megoldás elve: a mozgás fázisainak megfelelő számú (2 vagy 3 elég itt) sprite-ot kell alkotni, és felváltva tenni e sprite-okat a mozgás irányába mindig kissé előrébb. A filmek másodpercenkénti 24 képváltását el lehet érni. Természetesen ha több sprite-ot mozgat egy ciklus, úgy lassul a mozgás.

Javasoljuk a kedves olvasónak, hogy e fejezet áttanulmányozása és a programok kipróbálása után vágjon neki az előbb felvetett program megírásának.

```
100 print"<CLR>":rem"katica"
110 design 1,832
120 @...........
130 @..d......d..
140 @...d....d...
150 @....bbbb....
160 @...bbbbbb...
170 @...bbbbbb...
180 @b.cccccccc.b
190 @bbcccccccccbb
200 @..cccccccc..
210 @.ccbccccbcc.
220 @.ccbccccbcc.
230 @bbcccccccccbb
240 @b.cccccccc.b
250 @..cccccccc..
260 @..bccccccb..
270 @.b.cccccc.b.
280 @b...cccc...b
290 @....cccc....
300 @.....cc.....
310 @...........
320 @...........
350 cmob 0,9
360 mob set 0,13,2,0,1
370 mmob 0,150,200,150,100,3,200
```

Összefoglalóan elmondhatjuk, hogy a sprite-grafika a C64 olyan lehetősége, ami nagyon hiányzik a többi Commodore gépből. Igaz, hogy első látásra nem tűnik könnyűnek a sprite-ok programozása, de bizunk benne, hogy a fejezet elolvasása és példáinak kipróbálása után már készülnek a sprite-okat felhasználó szimulációs programok. A könnyen beszerezhető Simon's Basic segédprogrammal a csak BASIC nyelvet ismerő, kevéssé gyakorlott programirónak is lehetősége van szép és mozgalmas játékprogramokat készíteni és a tanitás szolgálatába állítani.

8.3. Színek a programokban

A színes televiziók olyan minőségi többletet jelentenek a fekete-fehérrel szemben, hogy az átlagban négyszeres ár ellenére is vezetik az eladási statisztikát.

A színek létrehozása az additiv színkeverés elvén alapul: a televiziós rendszerekben a piros, zöld és kék alapszínek intenzitását külön-külön veszik fel a kamerával és három elektronágyúval külön-külön hoznak létre minden képpont színének megfelelő keverési arányt a vevőkészülék képcsövében. A színeket is kezelő mikroszámítógépek a program által meghatározott színeket a televiziós rendszerrel megegyezően állítják be a kimenő jelben, ezért bármilyen - CCIR normában is dolgozó - televiziós vevőkészülék használható displayként. A legtöbb számítógép két kimenettel rendelkezik: RF és VIDEO. A VIDEO jelű kimenet csak a videojelet adja, ide speciális monitort, illetve video-bemenettel is rendelkező TV-készüléket lehet csatlakoztatni. Az RF-jelű kimenet a vivőfrekvenciát modulálva ugyanolyan jelet ad, amit a TV-antennán kapunk, ezért a TV-készülék antennabemenetére csatlakoztatható. A két megoldás között igen jelentős a minőségi különbség a video-csatlakozás javára: színekben gazdag programnál különösen feltűnő. Hogy színes monitort kevesen használnak, annak oka a magas ár (színes tv készülék 2-3-szorosa), de az is, hogy a monitor nem használható TV-műsor nézésére...

A színek jelentősége a művészetekben sem volt mindig egyforma, a festészet történetében gyakori volt a hangsúlyeltolódás. A velencei reneszánsz nagy mestere, Giorgione - a színek költője - nyugalmas, mozdulatlanságot sugalló kompozicióinak legfontosabb eleme a színfolt, a látvány. A Giorgionet példaképüknek tekintő impresszionisták szemében a világ színfoltok, fénytünemények összessége; a formát csak tapasztalatunk súgja, szemünk a színárnyalatokat látja, nem érzi a teret véleményük szerint. Az impresszionista festmények kissé vázlatszerűek, az első benyomáson alapuló, gyorsan felrakott színfoltokból tevődnek össze csodálatosan eleven, derűs képekké. Számítógépes grafikával elvileg jól utánozhatnánk az impresszionistákat a kis képernyőre felvitt sokszínű pontokkal - a számítógépes grafika már művészeti irányzatként is jelentkezett.

Más azonban a művészet és az oktatás, más a vászonra festeni és a képernyőre rajzolni. Oktatóprogram készitésénél mértékletességet kell tanúsitani, a színeknek legyen didaktikai jelentőségük, ne a látványra törekedjünk. Szöveges programban legfeljebb egy-két élénk, a szövegtől eltérő szint használjunk a fontos kifejezések, képletek, vagy a hibák kiemelésére. Sok esetben szép, és feltűnő lehet az alap színezése. Nagyon fontos a felhasználó szemének kimélése, főleg gyerekeknél. Ne használjunk túl kontrasztos megoldást, csak ha többen, messziről nézik a programot. A C16 és plus/4 alap színbeállitása erre jó, de közelről fárasztó. A C64 világoskék-sötétkék beállitása kedvező közelről a szemnek, de messzebbről nem látszik tisztán. A program betöltése előtt be lehet állitani a karakterek színét a színbillentyűkkel, ez meg is marad, ha a program nem ir elő mást. A C16 és plus/4 keret és háttér szine könnyen állitható a COLOR utasitással. A C64-nél az 53280 cimre a keret, az 53281 cimre a háttér színét lehet POKE utasitással beirni. Programból PRINT"<színbillentyű>", vagy PRINT CHR$(<színkód>) utasitással változtathatjuk az irás színét betűnként, vagy szavanként is. Egy ilyen utasitás hatása mindaddig megmarad, mig nem módositjuk. A gépek szintárába a képernyőtárhoz hasonlóan POKE utasitásokkal bevihetjük a képernyő adott helyén lévő karakter kivánt színét. A szintár cime:

C16, plus/4 : 2048 - 3047

C64 : 55296 - 56295

A szintár címei a képernyőtárral megegyezően sorfolytonosan helyezkednek el.

A Commodore gépeken lehetőség van több színű karakterek alkotására és a karakterek alapszíneinek egyedi beállítására is. E speciális igényeket kielégítő megoldások leírása megtalálható az alábbi két könyvben:

Úry L.: Commodore 64, LSI ATSZ, Budapest 1985.
Úry L.: Commodore C-16, C-116, LSI ATSZ Budapest, 1986.

A színkontraszt jelensége TV-képernyőn különösen feltűnő : egy szín a környezete színétől függően másként jelenhet meg. Didaktikai hibát követhetünk el, ha egy jelentéstartalommal bíró szín egyszer pl. sárga, máskor kék alapon jelenik meg : a felhasználó különbözőnek fogja látni.

A továbbiakban két példát mutatunk, amelyekben a színeknek döntő jelentőségük van. A szimulációs programok olyan jelenségeket is szimulálhatnak, melyekben a színek változása, vagy állandósága lényegi kérdés. A már említett kémiai elemzési módszer, a titrálás az indikátor színének megváltozását használja fel egy kémiai reakció teljes lefolyásának jelzésére. Az alábbi kis program kirajzolja a büretát, alá a poharat, és bemutatja a pohárban lévő oldat folyamatos színváltozását (C16 és plus/4):

```
10 graphic4,1
20 color0,14,7
30 color1,1,0
40 color2,2,6
50 color3,3,3
60 color4,14,4
70 draw,118,25 to 120,25:draw,125,25 to 127,25
80 draw,118,25 to 120,25:draw,125,25 to 127,25
90 draw,120,25 to 120,95:draw,125,25 to 125,105
100 draw,119,96:draw,124,106
110 draw,119,96 to 110,96:draw,124,106 to110,106
120 draw,110,90 to 110,110:draw,110,110 to 108,115: draw,108,115 to 106,110
130 draw,106,110 to 106,90:draw,106,90 to 110,90
140 draw,109,90 to 109,81 to104,81 to 104,85 to 107,85
150 draw1 to107,90
160 draw,99 ,116:draw,100,116 to 100,150 to 116,150 to 116,116
170 draw,117,116:draw,120,30 to125,30
180 paint 2,122,40,1
190 draw,101,121 to 115,121
200 box 3,101,122,115,149,,1
210 for i=1 to 500:next i:color 3,3,4
220 for i=1 to 500:next i:color 3,3,5
230 for i=1 to 500:next i:color 3,3,6
240 for i=1 to 500:next i:color 3,14,7
250 getkey t$:graphic0
```

A 20-60-as sorok beállítják a keret, háttér, írás színét és a többszínű üzemmód két újabb színét. A 210-240-es sorok a COLOR utasítás lehetőségét használják ki: az eddig (3) paraméterrel megadott erős piros szint (50-es sor) egyre halványabbra, majd háttér színűre változtatják. Vigyázni kell, hogy ezt a (3)-as szint máshol ne használjuk, mert a COLOR utasítás mindenütt kicseréli! Ebben a programban csak a 200-as sorban

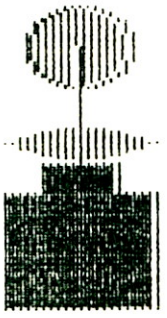
szerepel, a pohárban lévő oldatot színeztük ki egy BOX utasítással. Ugyanezt a feladatot a PAINT utasítás is elvégzi, de sokkal lassabban, míg a COLOR-ral való színcsere azonnali.

A színdinamika a színek pszichológiai, fiziológiai hatásaival, és a színek gyakorlati alkalmazásával foglalkozik, s bizonyos vonatkozásainak helye van az iskolai rajz-művészettörténet oktatásában. A mikroszámítógépek gazdag színskálája jó lehetőséget ad a színdinamika számára. Könnyen készíthetők a színlátást vizsgáló teszt-programok, építészeti, lakberendezési szintervezést segítő programok, stb. Az alábbi program a színdinamika iskolai tanításában használható programok készítéséhez ad ötletet. Kirajzol egy virágot, majd külön-külön kéri a virág, a váza, a levél és a háttér színét, színerejét. A C16-on futó program használhatja mind a 16 szint és színenként a 8 színerőt, mindezt négy terület színezésére: a lehetséges variációk száma 100 milliós nagyságrendű!

```
95 a=1:af=7:b=2:bf=7:c=13:cf=7:d=7:df=7:e=8:eh=7
100 rem"színek programja"
110 graphic 4,1
140 color 0,a,af
150 color 1,b,bf
160 color 2,c,cf
170 color 3,d,df
180 color 4,e,ef
200 draw 1,0,150 to159,150
210 draw 1,40,100 to40,130 to60,130 to 60,90 to 55,90 to55,80
220 draw 1,55,80 to 45,80 to 45,90 to40,90 to 40,100
240 rem"virág"
250 draw 1,50,80 to 50,40
260 circle 1,50,40,7
270 circle 1,50,40,19,,150,210 :circle 1,50,106 ,19,,330,30
290 paint3,50,100,1
300 paint 2,50,38,1
310 paint1,51,72,1:paint 1,49,72,1
320 char1,25,1,"1:fekete"
322 char1,25,2,"2:fehér"
324 char1,25,3,"3:piros"
326 char1,25,4,"4:encián"
328 char1,25,5,"5:bíbor"
330 char1,25,6,"6:zöld"
332 char1,25,7,"7:kék"
334 char1,25,8,"8:sárga"
336 char1,25,9,"9:narancs"
338 char1,25,10,"10:barna"
340 char1,25,11,"11:sárgászöld"
342 char1,25,12,"12:rózsaszín"
344 char1,25,13,"13:kékeszöld"
346 char1,25,15,"színerő: 1 - 7"
350 print"Játszunk tovább?  i/n"
355 getkey x$
358 if x$="n"then end
360 if x$="i"then 400
362 print"i vagy n betűvel kérem a választ!":goto 350
400 input"háttér szine:";a$: a=val(a$)
402 input"színerő:";af$: af=val(af$)
410 input"levél, szár szine:";b$ : b=val(b$)
```

```
420 input"szinerő:";bf$: b=val(b$)
430 input"virág szine:";c$:c=val(c$)
440 input"szinerő:";cf$:cf=val(cf$)
450 input"váza szine:";d$:d=val(d$)
460 input"szinerő:";cf$:cf=val(cf$)
480 print"Gombnyomásra indul!" :getkey t$:goto110
500 graphic0:end
```

A program a használható ötféle szint és a hat színerőt változók aktuális értékeiként tartja nyilván, mely értékeket INPUT-al kéri be a 400-460-as sorokban. A 95-ös sor megad egy kezdő színösszeállítást, melyet a 110-180-as sorok beállítanak. A 210-220-as sor a vázát, a 250-270-sor a virágot és levelét rajzolja, a 290-310-es sorok kiszinezik a virágot, a vázát és a levelet. A 320-346-os sorokkal kiirjuk a szükséges tudnivalókat. Sajnos többszínű üzemmódban torzulnak a betűk, ezért e kiiratást elhagyva és egy papirra leirva a szinek számait, talán esztétikusabb lesz a munka. A 350-362-es sorok a játék folytatása, illetve befejezése választást biztositják, utóbbi esetben lehet szineket választani, majd gombnyomással inditani az újrarajzolást. A programfutás egy ábrája:



Ilyen program könnyen készithető egy iskolai rajzfeladat, dekoráció, szinpadkép tervezéséhez, és természetesen kombinálható mozgásokkal is. Sprite-okat mozgató C64-programban a sprite-ok szinregisztereinek átirásával érhetünk el hasonló pillanatszerű szinváltásokat.

A TVC gépek a beállitott üzemmódnak megfelelően 2, 4, vagy 16 szint használhatnak. A keret, alap és irás szinét a C16-hoz hasonlóan külön kell beállitani:

SET PAPER szs : a karakterek alapszine (szs) szinsorszámnak megfelelő lesz.
SET INK szs : az irás szine (szs) szinsorszámnak megfelelő lesz.
SET BORDER pk : a képkeret szine a (pk) palettakódnak megfelelő lesz.

16 szinű üzemmódban az (szs) szinsorszám bármelyik lehet. 2 és 4 szinű üzemmódban a SET PALETTE utasitással kell kiválasztani a használni kivánt 2, illetve 4 szint, pl.:

SET PALETTE 0,65,68,84

négyszinű üzemmódban a fekete, kék, vörös és sárga szineket engedélyezi. Ebben az utasitásban a szinsorszámok palettakód-megfelelőjét kell használni (ld. gépkönyv). E beállitás után az engedélyezett 4 szin (szs) szinsorszámait használhatjuk a SET PAPER és a SET INK utasitásokban.

Zárt alakzatok kifestése is a C16-hoz hasonló módon történhet:

PLOT a,b,PAINT : az (a,b) koordinátájú pontot körülvevő zárt sikidomot festi a SET INK utasitással megadott szinűre.

Az iskolákban rendszerint kevés a szines TV-készülék és otthon sincs mindenkinek. Ezért a programkészitésnél feltétlenül gondolni kell a fekete-fehér készülékekre: olyan szineket használjunk, amelyek fekete-fehérben is megkülönböztethetők.

A **grafika** összefoglalásaképpen egy javaslat, amellyel a programokat végigpróbáló olvasó bizonyára egyetért:

Mozgatott alakzatokat karaktermozgatással érdemes létrehozni, módositva a karakterkészletet, illetve C64-en a Simon's Basic sprite-jaival. A GSHAPE-SSHAPE utasitáspár felhasználását azon olvasóinknak ajánljuk, akik nem kívánnak belemélyedni a karakterszerkesztés munkájába. A bemutatott egyszerű, kisméretű alakzatok igy is elfogadhatóan mozognak, nagyobb alakzatot több módositott karakter együtt mozgatásával érdemes létrehozni.

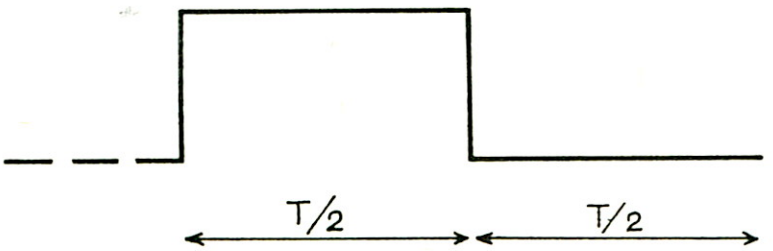
9. Hangos is lehet a program.

9.1. Hang előállítása

Csaknem minden mikroszámítógép rendelkezik valamilyen hangkeltési lehetőséggel. Az egyszerűbbek csak "csipognak", a C64 szinte teljes értékű szintetizátorként használható.

Játékprogramokban fontos a szimulált jelenség atmoszféráját képben és hangban egyaránt megteremteni (pl. futballmeccs, autóverseny). Oktatóprogramoknál azonban minden olyan tényezőre vigyázni kell, amelyik a szimulált jelenség megfigyelését, a kérdések megértését befolyásolhatja - ilyen tényező a hang is. Szöveges gyakoroltató programoknál legfeljebb a válasz minősitését kisérő dicsérő, illetve elmarasztaló kis dallam, hangeffektus javasolható. Ekkor is érdemes azonban a programnak megkérdeznie a felhasználót, hogy nem zavarja-e a hang, s a válasz alapján engedélyezni a hang megszólalását. Természetesen a TV-készülék hangerőszabályzója is használható. Program bevezetéseképpen, vagy jutalomként elképzelhető rövid, ismert dallamok lejátszása, egy program tanórai alkalmazását azonban a legkisebb hangeffektus is zavarhatja. Tanitási órán kivül inkább helye lehet a programban egy kis zenének. Más kategória az énekzene tanitását segitő program, itt a hang tartalmi kérdéssé válhat. Egyszerűen készithetünk adott frekvenciájú hangot megszólaltató programot, metronómot utánzó programot. A C64 alkalmas többféle hangszer szimulálására, segitheti a zeneszerzést.

Az elterjedt mikroszámitógépek között a PRIMO rendelkezik a legegyszerűbb hanggenerátorral. Programozása könnyű, de szolgáltatásai is szerények. A PRIMO hangja négyszögimpulzusok sorozata:



A programozást egyetlen utasitással végezhetjük:

BEEP a1,b1;a2,b2;a3,b3;...

Egy paraméterpár (a1,b1) megadása kötelező, a többi tetszőleges - minden paraméterpár egy hang megszólaltatását végzi. A paraméterpárok első tagja a hang magasságát (frekvenciáját), második tagja a megszólaltatott hang időtatartamát állitja be az alábbi képletek szerint:

$$a = \frac{T-70}{16.8} = \frac{(1/f * 10^{6}) - 70}{16.8} \qquad b = h*f$$

A képletekben (T) a periódusidő, (f) a frekvencia Hz-ben, (h) a hang hosszúsága másodpercben.

A PRIMO 32 - 1000 Hz frekvenciatartományban képes hangot kiadni, ez az (a) paraméter 1816 - 56 közötti értékeinek felel meg. A második paraméter (b) a periódusok számát jelenti, ezért azonos hosszúságú hangokhoz a frekvenciától függően más és más értékek tartoznak. E paraméter maximális értéke 32 767, ez a legkisebb frekvenciánál kb. 1000 másodpercet, a legmagasabbnál kb. 32 másodpercet jelent. Egy másodpercig szóló normál A hang keltése:

```
BEEP 131,440
```

Ahhoz, hogy a frekvencia biztosan pontos legyen, az úgynevezett NMI megszakításokat ki kell kapcsolni, ekkor a RESET gomb is hatástalan lesz:

```
POKE 16443,PEEK(16443) AND 127
```

Visszakapcsolás:

```
POKE 16443,PEEK(16443) OR 128:OUT 0,PEEK(16443)
```

Tulajdonképpen a 16443-as címen a 7.bitet kell 0-ra (kikapcsolás), illetve 1-re állítani (bekapcsolás).

A következő program egy tizbillentyűs zongorát szimulál, természetesen egyszerre csak egy billentyű "szólhat", nem úgy, mint az igazi zongoránál:

```
10 print"  BILLENTYŰZET : "
20 print"1 2 3 4 5 6 7 8 9 0 "
30 print:input"hanghosszúság (1-10):";h
40 a$=inkey$:if a$="" then 40
50 a=val(a$)
60 on a+1 goto 100,110,120,130,140,150,160,170,180,190
100 beep 56,98*h:goto 40
110 beep 1816,3*h:goto 40
120 beep 1280,4*h:goto 40
130 beep 900,6*h:goto 40
140 beep 600,9*h:goto 40
150 beep 440,14*h:goto 40
160 beep 280,20*h:goto 40
170 beep 200,29*h:goto 40
180 beep 130,44*h:goto 40
190 beep 90,62*h:goto 40
```

A hanghosszúság értékének 10-et adva kb. 1 másodperces hangot kapunk.

Az ilyen "zongora" - programot egy rövid hangeffektus készítésében lehet felhasználni: "zongorázással" megkeresni az alkalmas dallamot, majd a zongora-program megfelelő BEEP-paramétereit programunkba szubrutinként beépíteni.

Érdekes hanghatásokat általában ciklusban változtatott paraméterekkel lehet elérni. Az alábbi példa egy repülőgépmotor, vagy rakéta hangját utánozza:

```
10 for a= 1800 to 10 step -5
20 b=1000/a
30 beep a,b
40 next a
```

Gyakoroltató oktatóprogramokban használhatók az alábbi hangeffektusok. A 10-es sor a jó válasz dicséretére, a 20-as sor pedig a rossz válasz esetén, mintegy azt mondja: ejnye!

```
10 for a=0 to 3 step 0.1:beep 100*sin(a),10*a:next a
20 for a=20 to 100:beep a,a/10:next a
```

Az ilyen hangeffektusok nem tervezhetők pontosan előre, sok próbálkozással lehet eltalálni a megfelelőt.

A TVC gépek hangja hasonlóan programozható:

```
SOUND PITCH a, VOLUME b, DURATION c
```

(a) paraméter a hangmagasság (frekvencia) megadására szolgál az alábbi képlet szerint:

a = 4096 - 19531.5/f , ahol (f) a frekvencia Hz-ben.

A hangerőt megadó (b) paraméter értéke 0-15 között lehet, de elhagyható, alapértelmezese: 7.

Az időtartamot megadó (c) paraméter 20 msec lépésközzel állítható, azaz pl.:

c=1 : 20 msec, c=50 : 1 sec, c=6000 : 1 perc

Ez a paraméter is elhagyható, alapértelmezése: c=50 (1 sec).

A SOUND után nem írva írásjelet (mint fent), az új hang megszakitja a régit, míg pontosvesszőt írva befejeződik az előző hang az új megszólalása előtt.
1 másodpercig szóló normál 'A' hang keltése:

```
SOUND PITCH 3652
```

A PRIMO-nál látott zongora-program TVC-re:

```
10 rem"zongora TVC"
20 print"Billentyűzet:"
30 print" 1  2  3  4  5  6  7  8  9"
40 input prompt"Hanghossz (10-30):"h
50 get a$
60 a=val(a$)
70 if a$=chr$(32) then end
80 on a goto 110,120,130,140,150,160,170,180,190
110 sound pitch 300, duration h:goto 50
120 sound pitch 600, duration h:goto 50
130 sound pitch 900, duration h:goto 50
140 sound pitch 1200,duration h:goto 50
150 sound pitch 1700,duration h:goto 50
160 sound pitch 2100,duration h:goto 50
170 sound pitch 2500,duration h:goto 50
180 sound pitch 3000,duration h:goto 50
190 sound pitch 3500,duration h:goto 50
```

A rakéta hangja:

```
200 for a=100 to 4000 step 10
210 sound pitch a, duration 10
220 next a
```

A Commodore gépek igényesebb hanggenerátorral rendelkeznek. 2 egymástól független, egyidejüleg megszólaló hangot képes előállítani a C16 és a plus/4. E gépekben 5 TED-regiszter végzi a hangkeltés vezérlését, de a programozás könnyű, mert két BASIC utasitással állithatjuk e regisztereket:

VOL h : a hangerőt állítja be, h=0 esetén nem szól, h=7 a maximális hangerő.
SOUND a,b,c : az első paraméter a hangforrást választja ki. Lehetséges értékei:
1 : zenei hangforrás (négyszögjel)
2 : zenei hangforrás (négyszögjel)
3 : második (zaj) hangforrás

A második paraméter (b) értékével a hang magasságát (frekvenciáját) állíthatjuk be az alábbi képlet szerint:

b= 1024-(111840.45/f)

(f) a Hz-ben mért frekvenciát jelenti. Gyakorlatilag kb. 100-4000 Hz között lehet élvezhető hangokat kelteni (b=1023 lehet a legnagyobb érték).

A harmadik paraméter a hang megszólalásának idejét állítja be, értéke 65535-ig terjedhet. (c) értékének kiszámitása:

c = t*60 , ahol (t) a másodpercek száma.

Egy másodperces normál 'A' hang programja:

```
10 vol 5: sound 1,770,60
```

A PRIMOnál látott zongora-program, rakétahang és a két hangeffektus (jó és rossz válasz) programja C16-ra:

```
10 rem"zongora C16"
20 print"S":print:print"BILLENTYŰZET:"
30 print:print" 1 2 3 4 5 6 7 8 9 0"
40 print:input"Hangerő (0-7):";v$
50 print:input"Kitartási idő (1 - 50):";i$
60 v=val(v$):i=val(i$)
70 vol v
80 getkey h$:h=val(h$)
90 on h+1 goto 100,110,120,130,140,150,160,170,180,190
100 sound 1,900,i :goto 80
110 sound 1,10,i :goto 80
120 sound 1,50,i :goto 80
130 sound 1,100,i :goto 80
140 sound 1,200,i :goto 80
150 sound 1,300,i :goto 80
160 sound 1,400,i :goto 80
170 sound 1,500,i :goto 80
180 sound 1,700,i :goto 80
190 sound 1,800,i :goto 80
200 rem"rakéta"
210 vol4
220 for a=10 to 1023 step2
230 sound3,a,1
240 next a
250 end
```

```
260 rem"jó"
270 vol 4
280 for a=0.5 to 2.3step 0.05
290 sound2,1000*sin(a),2
300 nexta
310 end
320 rem"rossz"
330 vol4
340 for a=1000 to300 step-10
350 sound1,a,1
360 next a
```

A Commodore 64 zenei képességei kiemelkedőek, de teljes kihasználásuk komoly programozói munkát igényel. Ezért a fejezet további részét tekintse a kedves olvasó "kiegészitő anyagnak", kihagyása nem befolyásolja más fejezetek megértését. Elsősorban zeneértő fizikatanároknak, programot iró énektanároknak szólnak a továbbiak.

A hangok keltését a SID (Sound Interface Device) chip biztosítja. A SID 29 vezérlő regiszterrel rendelkezik, melyeket programból POKE utasitásokkal lehet beállitani a VIC-hez hasonlóan (ld. 8.2.3. fejezet). A SID szolgáltatásai magas zenei igényeket is képesek kielégiteni. A legfontosabb lehetőségek:

- három független hang
- 4 féle hullámforma
- burkoló görbe programozhatósága
- szinkronizáció, moduláció
- programozható szűrés

E sok lehetőség igen bonyolulttá teszi a programozást, itt úgy próbáljuk összefoglalni az alapokat, hogy bárki meg tudja szólaltatni gépét. A gépkönyvek a részleteket ugyanis nem magyarázzák el, ezért kezdő programozók nehezen boldogulnak e témával. Énekzene szakosok és zeneileg képzett olvasók részére ajánluk az alábbi könyvet:

Zenekönyv a Commodore 64-hez. DATA-BECKER-NOVOTRADE,1986.

A hangkeltés első lépéseként a hangerőt lehet beállitani a 24. regszterrel:

```
10 S=54272: POKE S+24,15
```

A báziscim (SID kezdőcime) megadása után maximális hangerőt állit be ez a sor (0-15 között lehet).

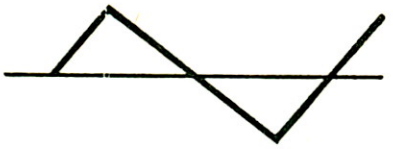
Következő lépésként célszerű a hangmagasság (frekvencia) beállitása. Mindhárom hang frekvenciáját két-két regiszterrel kell beállitani (alsó és felső byte):

1. hang alsó byte: 0. regiszter
1. hang felső byte: 1. regiszter
2. hang alsó byte: 7. regiszter
2. hang felső byte: 8. regiszter
3. hang alsó byte: 14.regiszter
3. hang felső byte: 15.regiszter

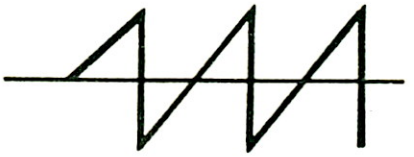
Az egyes frekvenciáknak, illetve zenei hangoknak megfelelő regiszterpárok értékeit a gépkönyvekben lévő táblázatból olvashatjuk ki. A normál 'A' hang (440 Hz) beállitása az 1. hangforrásra:

```
20 POKE S,69: POKE S+1,29
```

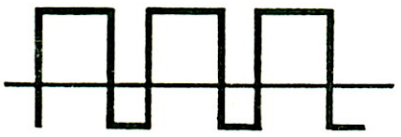
A SID négyféle hullámformát tud előállitani, ezek értékeit a 4. (1.hang), 11. (2.hang), illetve 18. (3.hang) regiszterbe kell beirni. E három regiszter 0. bitje segitségével lehet lágyan, kattanás nélkül kikapcsolni a hangot. Az alábbi ábrák mutatják a hullámformákat, az alattuk lévő szám a megfelelő regiszterekbe (4,11,18) irandó értéket adja meg. Kikapcsolás céljából mindegyik értéket eggyel kell csökkenteni (0.bit 0-ra állitása).



Háromszög-hullám
17



Fűrészfog-hullám
33



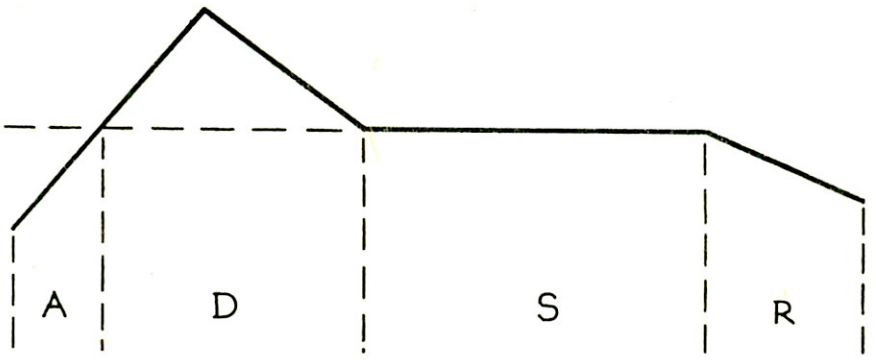
Négyszög-hullám
65

Fehér zaj
129

Az 1. hangforráson háromszög-hullámot ad az alábbi programsor:

```
30 POKES+4,17
```

A zenében igen fontos az egyes hangok hangerejének változása a megszólalástól elhalásig. Ezt az időtartamot négy részre osztva lehet programozni a burkológörbe-generátor (ADSR-generátor) segitségével. Az alábbi ábra érzékelteti a felfutási idő (A=attack time), lecsengési idő (D=decay time), kitartási idő (S=sustain time) és az elengedési idő (R=release time) fogalmát:



A burkoló görbe alakjának kialakitásával utánozhatjuk az egyes hangszerek jellemző hangját és létrehozhatunk bármilyen új hangzást is. A görbe mind a négy részének időtartamát 0-15 közötti paraméterekkel adhatjuk meg, melyek néhány millisecundumtól néhány secundumig terjedő időnek felelnek meg. Az AD szakasz után az S szakasz addig tart, amig az előzőekben emlitett módon ki nem kapcsoljuk az illető hangot a 4. regiszter 0. bitjének 0-ra állitásával:

```
50 FOR I=1 TO 50 :NEXT: POKE S+4,16
```

Miután ez megtörtént, akkor következik a megfelelő regiszterben megadott SR szakasz. Az ADSR-generátor regiszterei:

1. hang AD: 5. regiszter
1. hang SR: 6. regiszter
2. hang AD: 12. regiszter
2. hang SR: 13. regiszter
3. hang AD: 19. regiszter
3. hang SR: 20. regiszter

A 8 bites regiszterek alsó 4 bitje a D, illetve R, felső négy bitje az A, illetve S paraméterek értékeit kell tartalmazza (mind a négy paraméter 0-15 között lehet). A regiszterek programozása:

```
POKE S+5,A*16+D: POKE S+6,S*16+R
```

Ha az 'A' hangot zongoraszerűen akarjuk hallani, akkor gyors felfutást (A=0), lassú lecsengést (D=9) és gyors elengedést (R=0) állitsunk be. A zongorabillentyű nyomvatartási idejét jelenti az 50-es sorban megadott idő, ennek letelte után lép életbe a 6. regiszterben megadott S és R érték. Az elmondottak programja:

```
40 POKE S+5,9:POKE S+6,240
```

Hullámformának négyszög-hullámot választva még két regiszter beállitására van szükség az impulzusszélesség megadására. E regiszterek:

1. hang, felső byte: 3. regiszter
   alsó byte: 2. regiszter
2. hang, felső byte: 10.regiszter
   alsó byte: 9. regiszter
3. hang, felső byte: 17.regiszter
   alsó byte: 16.regiszter

Az impulzusszélességet az impulzus pozitiv részének a teljes periódushoz való arányával kell megadni, az arány 0-tól 4095-ig változhat. E 12 bites egész szám magasabb helyiértékű 4 bitje adja a felső byte értékét (0-15), alacsonyabb helyiértékű 8 bitje pedig az alsó byte értékét (0-255). A mélyebben érdeklődők kedvéért álljon itt egy példa e 12 bites számra. A bináris-decimális átszámitást a karakterszerkesztésnél látott sémában mutatjuk be:

```
helyiértékek: 2048 1024 512 256*128 64 32 16 8 4 2 1
bitek:          0    1    1   1 * 1   1  1  1 1 1 1 1 = 2047
```

A 2047-e számérték kb. megfelel az 50 %-os (szabályos) négyszögjelnek. A felső byte értéke a magasabb helyiértékű (csillag előtti) 4 bit, a megszokott 8 bites számként irva:

```
128 64 32 16 8  4  2  1
 0  0  0  0  0  1  1  1  = 7
```

Az alsó byte értéke a további 8 bit (mind =1), azaz 255. Az 1. hangforráson tehát szabályos négyszögjelet kapunk az alábbi utasitásokkal:

```
POKE S+4,65:POKE S+2,255:POKE S+3,7
```

A C64 az eddigieken kivül még sok lehetőséget biztosit a hangprogramozásra. A SID háromféle szűrőként programozható: alul-, felül-, illetve sáváteresztő szűrőként. A szűrőkkel változtathatjuk a hangok felharmonikus struktúráját, igy jobban megközelithetjük

az egyes hangszerek hangját. További szolgáltatás a három hangforrás kimenő jeleinek szinkronizálása, illetve modulálása egymással. Ezeket a megoldásokatt kombinálva gyakorlatilag végtelen variációs lehetőséghez juthat a programozó. Harangzúgástól az ismert zeneművek új hangszereléséig, és soha nem hallott hangzásokig, mindent meg lehet csinálni. Az igényes zeneprogramozáshoz az említett szakkönyvön túl csak türelem és idő szükséges.

A PRIMO és C16 gépekre közölt 'zongora-program' megfelelője C64-re:

```
10 rem"zongora C64"
15 print"S"
20 print:print"      Billentyűk:"
30 print:print" 1 2 3 4 5 6 7 8 9 0 + - | "
40 print:input"hangerő:";a
70 s=54272
80 pokes+24,a
90 pokes+5,136
100 pokes+6,248
110 pokes+4,8
120 fori=1 to 50:next i:pokes+4,16
130 get h$:if h$="" then130
140 if h$="1" then pokes,207:pokes+1,34
150 if h$="2" then pokes,255:pokes+1,36
160 if h$="3" then pokes,18 :pokes+1,39
170 if h$="4" then pokes,101:pokes+1,41
180 if h$="5" then pokes,219:pokes+1,43
190 if h$="6" then pokes,118:pokes+1,46
200 if h$="7" then pokes,58 :pokes+1,49
210 if h$="8" then pokes,39 :pokes+1,52
220 if h$="9" then pokes,65 :pokes+1,55
230 if h$="0" then pokes,138:pokes+1,58
240 if h$="+" then pokes,5  :pokes+1,62
250 if h$="-" then pokes,181:pokes+1,65
260 if h$="|" then pokes,157:pokes+1,69
270 pokes+4,17:goto 120
290 rem"rakéta"
300 s=54272:print"S"
310 poke s+24,10
320 for a=1 to 40 :for b=1 to 255 step 5 *a
325 poke s+19,136:poke s+20,248
330 poke s+18,129
340 poke s+15,a:poke s+14,b
350 nextb:next a
360 poke s+18,128
380 stop
400 s=54272
410 print"jó!"
420 poke s+24,10
430 pokes+4,33
440 poke s+5,9:poke s+6,240
450 for a=0.1 to 1.2  step 0.1
460 poke s,10*a:poke s+1,100*sin(a)
465 for z=1 to 2 :next
470 next a
```

```
490 poke s+4,32
500 stop
600 s=54272:print"rossz!"
610 for a= 90  to 2 step -4
620 poke s+5,136 :poke s+6,248
630 poke s+4 ,33
640 poke s,a:poke s+1,a
650 for z= 1 to 8   :next z
660 poke s+4 ,32
670 next a
```

A program az 1. hangforrást használja. A 80-as sor a hangerőt, a 90-100-as sorok a burkológörbét, a 270-es sor a hullámformát állitja be a billentyűnyomás után. A 120-as sor kitartja a hangot, majd kikapcsolja. A 140-260-as sorok a billentyűnyomás alapján a frekvencia beállitását végzik végtelen ciklusban. A zongorázást leállitani <stop> billentyűvel lehet.

A 300 - 670- es sorok az eddig látott rakéta-indulás, illetve a jó és rossz válaszra adható hangeffektus egy lehetőségét mutatják be. Mindenképpen érdemes felhasználás előtt e minta listákat izlés szerint módositani.

9.2. Programok összefűzése

A hangeffektusok általában nem lényegi részei egy oktatóprogramnak – így elkészítésük is többnyire a munka végére marad. Egy-egy jól bevált dallam viszont több programban felhasználható, nem kell mindig újat kitalálni. Más jellegű programrészek, pl. új karakterek, kérdező rutinok, rajzok szintén alkalmasak lehetnek több programban való felhasználásra. Természetesen könnyebb egy kész listát bepötyögni a gépbe, mint megírni a megfelelő programrészt. Legkönnyebb azonban a már egyszer megírt és újra felhasználható programrészeket összefűzni, általánosan használható rutinokat kifejleszteni és az egyes programokba szubrutinként beépíteni. A gyakran meghívandó szubrutinokat legjobb a program elején elhelyezni. Egy szubrutin híváskor ugyanis a gép a legkisebb sorszámtól kezdve keresi a szubrutint – így hamarabb megtalálja.

Az eddig vázoltak megvalósítását – programok összefűzését – a legtöbb gép BASIC nyelve nem segíti alkalmas utasítással. A feladat megoldása azonban nem nehéz, csak a program tárbeli elhelyezéséről kell egy keveset tudni. A Commodore gépek a program után helyezik el a változókat. A program, változók és egyéb, a belső szervezés miatt fontos adatok tárbeli címeit (mutatók) a memória alsó 2 Kbyte-ja tartalmazza, C16, Plus/4 és C64 gépeknél is azonosan:

BASIC-program kezdőcíme,	alsó byte:	43 ($2B)
	felső byte:	44 ($2C)
Változóterület kezdete,	alsó byte:	45 ($2D)
	felső byte:	46 ($2E)
Tömbváltozók kezdete,	alsó byte:	47 ($2F)
	felső byte:	48 ($30)
Első szabad byte címe,	alsó byte:	49 ($31)
	felső byte:	50 ($32)

E néhány mutató ismeretében könnyen fűzhetünk össze programokat. Ha a gépbe betöltünk, elindítunk, vagy listáztatunk egy programot, a gép a 43 és 44 címekből veszi a program kezdetét. E címek tartalma azonban BASIC-programból, vagy parancsból módosítható. Az összefűzés elve:

- A hozzáfűzendő programot átsorszámozni úgy, hogy a legkisebb sorszám is nagyobb legyen az első program legmagasabb sorszámánál.
- Az első programot betölteni.
- A program kezdetét jelző mutatókat átállítani a program végére.
- Betölteni a második programot.
- Visszaállítani a program kezdetét jelző mutatókat az eredeti értékre.

A program átsorszámozását a C16 és Plus/4 gépek a RENUMBER paranccsal elvégzik:

RENUMBER a,b,c : (c) sorszámtól kezdve átsorszámoz, az új sorszámok (a)-val kezdődnek, a soronkénti növekmény (b). Ha egyik paramétert sem írjuk ki, akkor az egész programot sorszámozza át 10-el kezdve 10-es növekménnyel. Ha véletlenül egy ugró utasításban olyan sorszám szerepel, amilyen a programban nincs – futtatás közben nem feltétlenül kapunk hibajelzést – RENUMBER parancs esetén a program elszállhat. Emiatt átsorszámozás előtt mindenképpen tároljuk a programot.

A C16 és a Plus/4 felülről kompatibilis a C64-el – azaz a C64 BASIC minden utasítását ismeri, így a C64-programok is átsorszámozhatók a C16-on, vagy Plus/4-en. A C64 BASIC-programok futtathatók is a másik kettőn, amennyiben nem tartalmaznak POKE utasítást (a tárbeli címek eltérnek!).

Az első program betöltése után PRINT PEEK(45) és PRINT PEEK(46) parancsokkal kiolvashatjuk a változóterület kezdőcimét. Ennek értékét kettővel csökkentve kapjuk a program végének cimét, amit a 43 és 44 cimekre kell beirni. A 45-ös cimen van az alsó byte (alacsonyabb helyiértékű byte), ezért ennek számértékét kell 2-vel csökkenteni. Ha ennek az értéke 0, vagy 1, akkor csökkentve 254, vagy 255 lesz, de ekkor a felső byte (46-os cim) értékét is eggyel csökkenteni kell. A csökkentéssel kapott értékeket POKE parancsokkal lehet betölteni a 43, 44 cimekre.

A második programot a gép a módositott cimtől kezdi tölteni, igy folyamatosan kapcsolódik a már tárban lévő programhoz.

A mutatók visszaállitása azt jelenti, hogy a 43 és 44 cimekre az eredetileg ott lévő értékeket irjuk vissza - ezt érdemes a művelet előtt feljegyezni. Alapesetben ezek az értékek:

C16, Plus/4:	PEEK(43)	= 1
	PEEK(44)	= 16
C64:	PEEK(43)	= 1
	PEEK(44)	= 8

A PRIMO gépek sem sokban térnek el. A programok tárbeli elhelyezése hasonló, ezért az összefűzés az eddig leirtakkal teljesen azonosan végezhető, csak a cimek különböznek:

BASIC program kezdőcime,	alsó byte:	16548
	felső byte:	16549
Változóterület kezdőcime,	alsó byte:	16633
	felső byte:	16634

Mivel a PRIMO gépek nem ismerik a RENUMBER parancsot, ezért a sorszámozási problémák kikerülésének legegyszerűbb két lehetősége:

- Minden program irását 1000-es sorszámmal kezdeni, ekkor a kis sorszámmal irt szubrutinok elférnek előtte.
- A szubrutinokat egyenként különböző magas sorszámokkal irni (50000 felett), igy bármely program után elhelyezhetők.

A Simon's Basic egyszerű lehetőséget ad a C64-en programok összefűzésére:

MERGE <név>,ksz : a <név> programot a (ksz) számú készülékről (1: magnó, 8:floppy) hozzáfűzi a gépben lévőhöz.

Minden program-összefűzésnél két dologra nagyon ügyeljünk:

- A betöltendő program minden sorszáma nagyobb legyen a tárban lévő program legmagasabb sorszámánál!
- A két összefűzendő programban ne legyenek azonos változók! Ennek az eltévesztése nem okoz feltétlenül hibát, de ha igen, akkor nehéz felfedezni és kijavitani.

Gyakorlott programozók sokszor az egész programot szubrutinokból állitják össze. Oktatóprogramnál ez az eljárás kevéssé követhető, de hasznositható elemeit érdemes átvenni. Az első néhány program elkészitése után mindenkinek kialakul bizonyos programozói módszere, stilusa. Tulajdonképpen minden programnál csak a felhasználó által láthatók döntik el értékét, nem a program listája, belső szerkezete - ezért mondhatjuk, hogy utóbbi a programiró magánügye. Ennek ellenére érdemes a gyakorlottabb programozók fogásait megismerni, ötleteket meriteni programjaikból.

10. Játék, verseny, oktatás

10.1. Szaktárgyi játékok

"Az ember valamennyi állapota közül épp a játék és csak a játék az, ami teljessé teszi őt" - írta Schiller 1793-ban. Ma az ügyesebbnél ügyesebb gyermekjátékok (modellek, lego,stb.) néha jobban vonzzák a szülőket, a szerencsejátékok népszerűek - mindenki játszik valamilyen formában. A kisgyermek fejlődésében nélkülözhetetlen szerepe van a játéknak - a tárgyaknak és a cselekvésnek egyaránt. Az iskolának nem szabad a játékot és a tanulást szembeállítania - ebben csak az utóbbi veszíthet! Az alsó tagozatban még sokszor játékos a tanulás, a felső tagozat és a középiskola komolyabb tanulmányait is segítheti a játék, a versengés.

A magyar nyelv a 'játék' szót kétféle értelemben használja: az angol 'play' és 'game' megfelelőjeként is. A 'play' folyamatos tevékenység általában, míg a 'game' egy konkrét véges játékszakasz, magyarul játszmának jobb fordítani. Pedagógiai szempontból mindkét értelmezés fontos. A játékos tevékenység kezdetben csak öncélú, később a motiváció szempontjából is jelentőssé válhat. A játszma, mint küzdelem, verseny erős nevelő hatású, a problémamegoldó szellemi küzdelem az értelmi fejlődés előmozdítója.

A számítógépes játékok az utóbbi években meghódították a fiatalokat és kevésbé fiatalokat. A sok öncélú lövöldözős játék mellett nem kevés a hasznos, reflexeket, stratégia kialakítást, gondolkodást fejlesztő játék. Ezért semmiképpen nem szabad kategórikusan elvetni az ismert játékprogramokat, az iskolában is lehet helye egy autóverseny, futball, stb. programnak.

Az oktatás számára legnagyobb lehetőséget a szaktárgyak iránti érdeklődést felkeltő, egyes alkalmazásokat gyakorló speciális játékok jelentik, melyekből sajnos kevés van. E fejezetben ilyen játékok készítésére kívánjuk motiválni az olvasót.

Az igazán látványos gyorsan mozgó figurákat használó programok gépi kódban készülnek. Az a tény, hogy a gép értelmező programjának (BASIC interpreter) nem kell soronként saját nyelvére fordítani a BASIC-sorokat, többszázszoros sebességnövekedést eredményez. A gépi kódú programozás lényegesen több ismeretet kíván, mint a BASIC, de jelentősebb matematikai előképzettség nélkül is megtanulható (ld. ajánló bibliográfia). A C64-en könnyű összehasonlítani a két módszert. Próbáljuk meg ugyanazt a sprite-ot előállítani és mozgatni BASIC-programmal és a Simon's Basic segédprogrammal (ez gépi kódú rutinokkal mozgat). BASIC-programban úgy gyorsíthatók a mozgások, hogy a ciklusváltozó értéke nem 1-el, hanem nagyobb számmal növekszik. Természetesen így a mozgás nem lesz olyan folyamatos, de ez nem mindig baj. A 'HCI' című program érzékelteti, hogy ebből még előnyt is lehet kovácsolni: a molekulák ugráló mozgását utánozni.

Versengést lehetővé tevő programokban az esélyegyenlőség megköveteli, hogy bizonyos kiinduló adatokat véletlenszerűen adjunk, mint kártyában a lapot. Minden számítógép rendelkezik úgynevezett véletlenszám-generátorral, amelyik számelméleti függvények felhasználásával állít elő véletlen számokat. A megfelelő utasítás a Commodore és PRIMO gépeken azonosan használható:

RND(X): 0 és 1 közé eső véletlen számot állít elő. Leggyakrabban egy adott intervallumban van szükségünk véletlen számra:

A=100 + 100*RND(1) : hatására (A) 100 és 200 közé eső véletlen szám lesz. Hogy kapott számunk minél 'véletlenebb' legyen, az alábbi módszert javasoljuk:

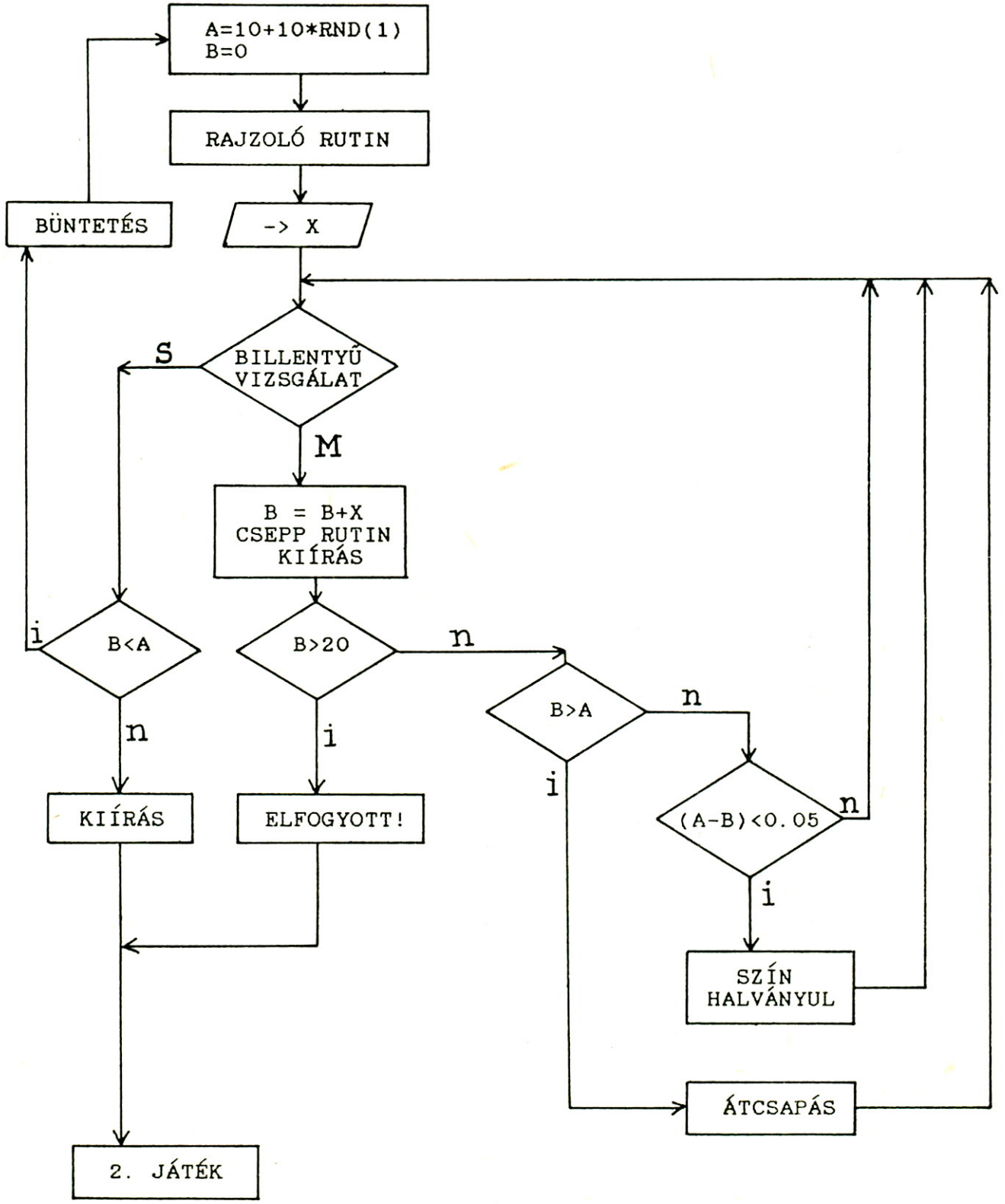
```
10 A=RND(0)
20 A=100+100*RND(1)
```

A továbbiakban két olyan játékot mutatunk be részletesebben, amelyek ugyan a kémia tantárgy népszerűsitését szolgálják, de szerkezetük és megoldásaik könnyen átültethetők más témára.

A már emlitett 'Titrálás' cimű játék lényege: véletlenszerűen adott mennyiségű oldatot kell ismert oldattal "megtitrálni", a program a pontosságot és gyorsaságot értékeli. A a szükséges tudnivalók után felhasználói kérésre kezdődik a játék (ld. folyamatábra):

- Véletlenszerűen adja a titrálandó mennyiséget, nullázza az időt, és a mérőoldat mennyiségét: 1160–1180–as sorok.
- Kirajzolja a poharat és a bürettát, közben elsötétitve a képernyőt: 2350–2600–as sorok.
- Bekéri a mérőoldat egy gombnyomásra eső mennyiségét (X):1230–1240–es sorok.
- Megvizsgálja a billentyűzetet : 'S' (=stop), vagy 'M' (=még) billentyű lett–e lenyomva, ha más, akkor az 1320–as sor visszaküld a billentyűvizsgálatra – azaz más betűk hatástalanok: 1260, 1280 és 1300–as sorok.
- 'S' betű esetén megvizsgálja, hogy a hozzáengedett mérőoldat (B) elérte–e a pohárban lévő (A) mennyiséget: ha igen, továbbküld a második játékra, ha nem, akkor újra kezdeti a játékot, de az eddigi idő kárbaveszett: 1340–1350–es sorok.
- 'M' betű esetén megnöveli B értékét, meghivja a cseppentést rajzoló rutint, kiirja B új értékét: 128–1290–es sorok.
- Megvizsgálja, van–e még mérőoldatunk, B nem nagyobb–e 20–nál. Ha igen, akkor kiirja és a második játékra küld: 1310–es sor.
- Ha van még mérőoldat, akkor megvizsgálja, hogy a hozzáengedett mérőoldat mennyisége (B) elérte–e már a meghatározandó oldat mennyiségét (A), ha igen, akkor szinátcsapást mutat és visszatér a billentyűvizsgálatra: 1330–as sor.
- Ha még nem érte el A értékét, akkor megvizsgálja, hogy megközelitette–e 0.05–nál jobban. Ha igen akkor szint halványit és visszatér a billentyűvizsgálatra: 1315–1320–as sorok.

A program azzal is utánozza a valódi titrálást, hogy figyelmetlen munka esetén lehet túltitrálni, azaz átcsapás után is engedi a további hozzácsepegtetést, amig el nem fogy a mérőoldat. Az alábbi lista C16, illetve Plus/4 gépen fut, a rajzoló rutinok kivételével C64–en is. A blokkséma alapján azonban bármilyen gépre könnyen átirható.


A=10+10*RND(1)
B=0
RAJZOLÓ RUTIN
-> X
BÜNTETÉS
BILLENTYŰ VIZSGÁLAT
S
M
B = B+X
CSEPP RUTIN
KIÍRÁS
B<A
i
n
KIÍRÁS
B>20
n
i
ELFOGYOTT!
B>A
n
i
(A-B)<0.05
n
i
SZÍN HALVÁNYUL
ÁTCSAPÁS
2. JÁTÉK

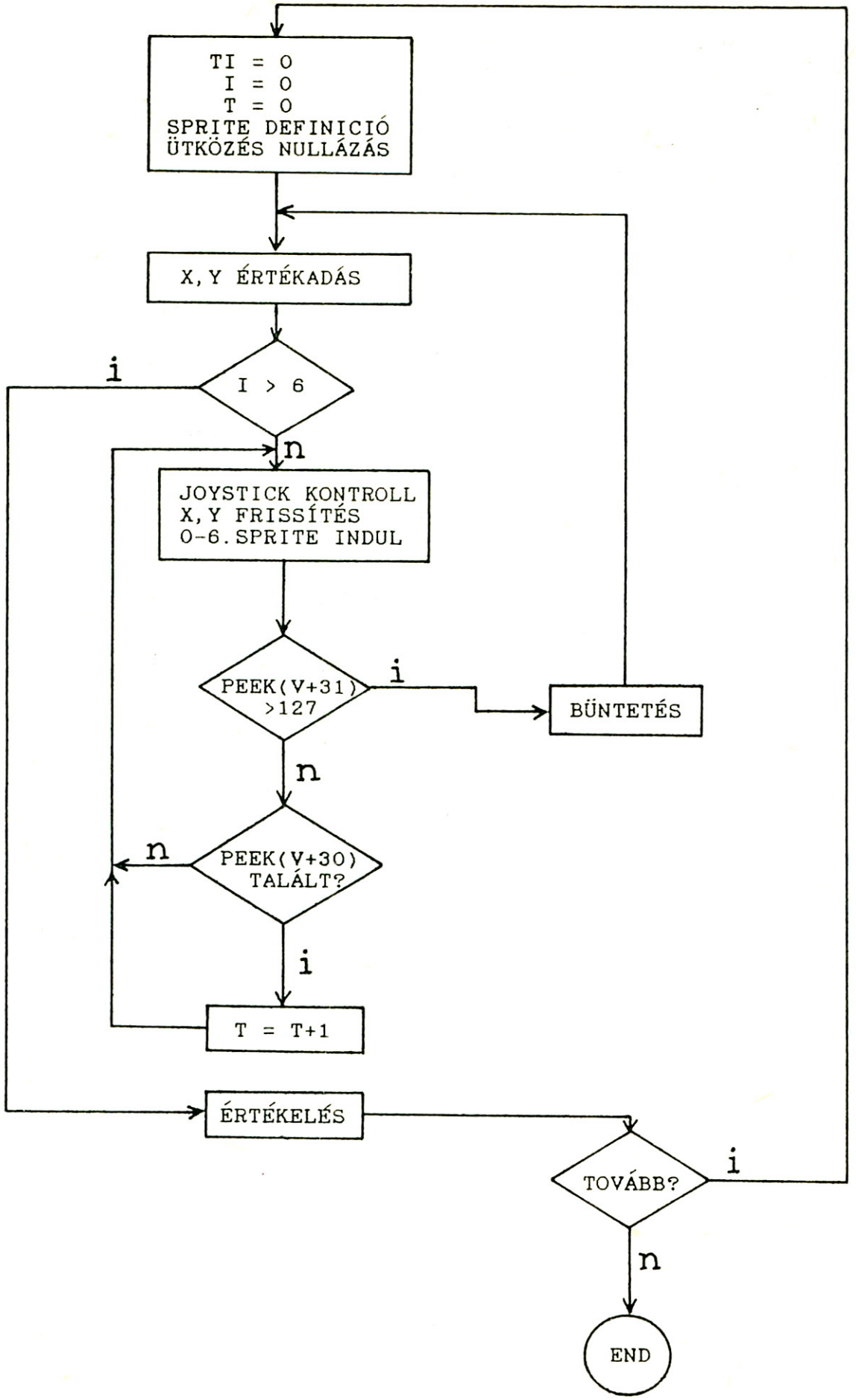
```
1160 ti$="000000":v=rnd(1):v=0
1170 v=10+10*rnd(1):b=0:print"S"
1180 a=int(v*100)/100
1200 gosub 2590
1210 gosub 2350
1220 gosub 2600
1230 input" 1 gombnyomas=";x$:x=val(x$)
1240 if x<=0 then print"Butasag!":goto 1230
1250 print"Titralj!"
1260 get q$
1270 if q$="" then 1260
1280 if q$="m" then b=b+x:gosub 2300
1290 printchr$(145)"                    "b"          "
1300 if q$="s" then 1340
1310 if b>20then print"Elfogyott!":goto1380
1315 if(a-b)<0.05 then color3,3,6
1320 if b<a then 1260
1330 color3,14,7     :goto 1260
1340 if b<a then 1360
1350 if b>a then 1380
1360 print"Atcsapas elott lealltal! Ujra kezded!"
1370 for i=1 to 1000:next i:goto 1170
1380 print"Masodik titralasod kovetkezik.  Tovabb?"

2300 draw 1,108,117:draw 1,108,118
2310 for i=1 to 50:next
2320 draw 0,108,117:draw 0,108,118
2330 if b>20 then b=21
2340 box 0,121,30,124,29+int(b)*3,,1:return
2350 graphic 4,1
2360 color0,14,7
2380 color1,1,0
2390 color2,2,6
2400 color3,3,4
2410 color4,14,4
2420 draw,118,25 to 120,25:draw,125,25 to 127,25
2430 draw,118,25 to 120,25:draw,125,25 to 127,25
2440 draw,120,25 to 120,95:draw,125,25 to 125,105
2450 draw,119,96:draw,124,106
2460 draw,119,96 to 110,96:draw,124,106 to110,106
2470 draw,110,90 to 110,110:draw,110,110 to 108,115:draw 1,108,115 to 106,110
2480 draw,106,110 to 106,90:draw,106,90 to 110,90
2490 draw,109,90 to 109,81:draw 1 to104,81:draw 1 to 104,85:draw 1 to 107,85
2500 draw 1 to 107,90
2510 draw,99 ,116:draw,100,116 to 100,150:draw 1 to 116,150:draw 1 to 116,116
2520 draw,117,116:draw,120,30 to125,30
2530 paint 2,122,40,1
2540 draw,101,121 to 115,121
2550 box 3,101,122,115,149,,1
2560 char 1,3,4,"m: meg":char 1,7,3,",":char 1,3,6,"s: stop"
2570 char 1,3,8,"tizedespont:0.05 !"
2580 return
2590 k=dec("ff06"):poke k,peek(k) and 239:return
2600 k=dec("ff06"):poke k,peek(k) or 16 :return
```

A 'HCl' nevű programban (C64) joystickkel vezérelhető HCl-molekulával kell minél többet eltalálni a képernyőn lassan 'átdiffundáló' ammónia-molekulák közül. A program a találatok számát és a gyorsaságot értékeli. Hat játszmát engedélyez, mindegyikben máshová helyezett 'adszorbensek' veszélyeztetik a 'vadászó' HCl-molekulát: ütközve elnyelik (egy játszma elvész). A mozgás sebességét a felhasználó adhatja meg (K), értéke 5 és 10 között lehet. A program folyamatábrája valamivel egyszerűbb az előzőnél, de a lista sokkal hosszabb a sprite-ok, illetve a joystick miatt. A közölt lista jelentősen rövidíthető ON...GOTO és DATA utasítások felhasználásával, de a könnyebb érthetőség miatt ezt nem tettük meg.

A program a szükséges tudnivalók közlése után felhasználói kérésre indul:

- Bekéri a kívánt sebességet: 250-330-as sorok.
- Időt, játszmák számát, találatok számát nullázza, sprite-okat definiálja: 360-520-as sorok.
- Megvizsgálja a játszmák számát, ha nagyobb 6-nál, a sprite-okat letiltja, értékelésre küld, különben továbbenged: 470-es sor.
- Elindítja a ciklust: 530-as sor (940-ig).
- HCl-sprite induló koordinátáit megadja véletlen ingadozással:550-es sor
- A joystick állása alapján frissíti az X,Y koordinátákat: 560-760-as sorok.
- A többi sprite koordinátáinak értéket ad: 790-850-es sorok.
- Lekérdezi a 7. sprite (HCl) és a háttér ütközését: 860-as sor, ha nincs ütközés, továbbenged.
- Ha történt ütközés, büntet: időveszteség, visszaküldés:1040-1050-es sorok.
- Lekérdezi a 7. sprite másik sprite-okkal való ütközését, ha nincs, mehet tovább a ciklus: 870-940-es sorok.
- Ha van ütközés, a "lelőtt" sprite-ot pillanatra letiltja, a találatok számát növeli és folytatja a ciklust (a lelőtt sprite-ot a következő pozícióban újra engedélyezi):950-1010-es sorok.

A folyamatábra elemeit természetesen a programban másképpen is meg lehet fogalmazni. Ugyanakkor a folyamatábra tartalmát bármely mikroszámítógépen megvalósíthatjuk, ha a sprite-okat egy-egy karakterrel (vagy több együttmozgatottal, módosítottal) helyettesítjük. Egy óvodásoknak való játék: a 7. sprite legyen a 'pí' betű, mint kutya, a többi sprite pedig 'Ö' betű, mint macska!


TI = 0
I = 0
T = 0
SPRITE DEFINICIÓ
ÜTKÖZÉS NULLÁZÁS
X,Y ÉRTÉKADÁS
I > 6
i
n
JOYSTICK KONTROLL
X,Y FRISSÍTÉS
0-6.SPRITE INDUL
PEEK(V+31)
>127
i
BÜNTETÉS
n
PEEK(V+30)
TALÁLT?
n
i
T = T+1
ÉRTÉKELÉS
TOVÁBB?
i
n
END

```
10 print"S"
20 print"Ammonium-klorid eloallitas"
30 print"**************************"
40 print
50 print"A jatek celja:minel gyorsabban"
60 print"minel tobb ammonium-kloridot keszitenl."
70 print
80 print"A joystick segitsegevel iranyithatja"
90 print"a hcl-molekulat, a tuzelogombbal se-"
100 print"besseget megduplazhatja."
110 print"Vigyazzon, adszorbenssel utkozve egy"
120 print"jatekot veszit!"
130 print
140 print"Nem minden utkozes hatasos, telibe "
150 print"kell talalni."
160 print
170 print"Spontan diffuzio is fellep, ezert"
180 print"a joystick hasznalata nelkul is mozog"
190 print"a hcl-molekula"
200 print
210 print"Egy betut megnyomva indul a jatek"
220 get a$ :if a$="" goto 220
230 for c=10to 220 step5
240 print"S":c=0:x=0:y=0:k=0
250 print" Milyen gyorsan akar jatszani?"
260 print" Lehetseges sebessegek 5-tol 10-ig."
270 print" Irja be a valasztott szamot!"
280 print" utana: [return]"
290 input k :print"S"
300 if k<5 or k>10 then 330
320 goto 360
330 print"5 es 10 kozotti szamot kerek!":goto 290
360 ti$="000000"
370 i=0:t=0
390 v=53248
400 pokev+21,255
410 poke 2047,13
420 poke 2041,14:poke2042,14:poke2043,14:poke2044,14
430 poke 2045,14:poke2046,14:poke2040,14
440 for n=0 to 62:readq:poke832+n,q:next
450 for m=0 to 62:readq:poke896+m,q:next
455 pokev+39,7:pokev+40,7:pokev+41,7:pokev+42,7 :pokev+43,7
457 pokev+44,7:pokev+45,7:pokev+46,1
460 pokev+30,0:pokev+31,0
470 x=200*rnd(1) :y=210:i=i+1:if i>6 then1070
480 poke1110+10*rnd(1),102:poke1225+20*rnd(1),102
490 poke1825+20*rnd(1),102:poke1550+15*rnd(1),102
500 for s=1050 to 2010 step 40
510 poke s,93
520 next s
530 for c=20 to 220 step 5
540 a=peek(56320)
550 x=x+4 *rnd(1):y=y+5*rnd(1)
560 if a=119 then x=x+k
```

```
570 if a=118 then x=x+k:y=y-k
580 if a=103 then x=x+2*k
590 if a=102 then x=x+2*k:y=y-2*k
600 if a=123 then x=x-k
610 if a=122 then x=x-k:y=y-k
620 if a=107 then x=x-2*k
630 if a=106 then x=x-2*k:y=y-2*k
640 if a=125 then y=y+k
650 if a=121 then x=x-k:y=y+k
660 if a=109 then y=y+2*k
670 if a=105 then x=x-2*k:y=y+2*k
680 if a=126 then y=y-k
690 if a=117 then x=x+k:y=y+k
700 if a=110 then y=y-2*k
710 if a=101 then x=x+2*k:y=y+2*k
720 if x>250 then 470
730 if x<4 then 470
740 if y>250 then 470
750 if y<1 then 470
760 pokev+14,x:pokev+15,y
790 pokev+2,rnd(1)*10+10:pokev+3,c
800 pokev+4,rnd(1)*10+40:pokev+5,0.8*c
810 pokev+6,rnd(1)*5+80:pokev+7,0.9*c
820 pokev+8,rnd(1)*8+120  :pokev+9,c-10
830 pokev+10,222-c :pokev+11,50+rnd(1)*10
840 pokev+12,225-c/2  :pokev+13,100+rnd(1)*10-c/4
850 pokev,c:pokev+1,0.8*c
860 if peek(v+31)>127 then goto 1040
870 if peek(v+30)=129 then950
880 if peek(v+30)=130 then 960
890 if peek(v+30)=132 then 970
900 if peek(v+30)=136 then 980
910 if peek(v+30)=144 then 990
920 if peek(v+30)=160 then 1000
930 if peek(v+30)=192 then 1010
940 next c :goto 470
950 pokev+21,254:t=t+1:print,,t:next c:goto470
960 pokev+21,253:t=t+1:print,,t:nextc:goto470
970 pokev+21,251:t=t+1:print,,t:nextc:goto470
980 pokev+21,247:t=t+1:print,,t:nextc:goto470
990 pokev+21,239:t=t+1:print,,t:nextc:goto470
1000 pokev+21,223:t=t+1:print,,t:nextc:goto470
1010 pokev+21,191:t=t+1:print,,t:nextc:goto470
1040 print"S":print"elnyelt az adszorbens!":forp=1 to 500:next:print"S"
1050 goto 470
1070 print :poke v+21,0:
1080 print"Letelt a jatek,vege!"
1090 print"Eloallitott"int(ti/60)" sec alatt"
1100 print
1110 print t;" mol ammonium-kloridot."
1120 if ti/60<50 then print"Gyors munka volt!":goto1160
1130 if ti/60>120 then print"Lassan jatszott.":goto 1160
1140 if ti/60>80 thenprint"Lehetett volna kicsit gyorsabb.":goto 1160
1150 print"Eleg gyors volt   "
```

```
1160 if t>10 then print"Szep mennyiseget osszehozott!"  :goto 1200
1180 if t<6 then print"Eleg keveset tudott osszehozni!" :goto 1200
1190 if t>6 then print"Kozepes volt a termelese.":goto 1200
1200 os=10*t+(80-ti/60)
1210 print
1220 print"Eredmenye: ";int(os);" pont"
1230 if os>180then print"Gratulalok!":goto1290
1240 if os<180then 1250
1250 if os>100then print"Eleg jo":goto 1290
1260 if os<100then 1270
1270 if os>60 then print"Nem valami fenyes!":goto 1290
1280 if os<60 then print"Ezt meg gyakorolni kell!":goto 1290
1290 print"Jegyezze fel az eredmenyt!"
1300 print"Keszulhet a kovetkezo versenyzo,"
1310 print"Egy betu lenyomasaval indul ujra!"
1320 get a$: if a$="" goto 1320
1325 goto 10
1330 data 0,0,0,0,0,0,0,0,0,0,0,0
1340 data 0,0,0,7,255,192,15,255,224,11,177,176
1350 data 27,182,184,24,55,184,27,183,184,27,182,184
1360 data 11,177,144,15,255,240,7,255,224,0,0,0
1370 data 0,0,0,0,0,0,0,0,0,0,0,0,0,0
1380 data 0,0,0,0,0,0,0,62,0,1,193,192
1390 data 6,0,48,8,0,8,11,36,68,19,36,66
1400 data 18,164,66,18,167,210,18,100,74,18,100,90
1410 data 18,36,74,8,0,18,7,0,4,0,224,120
1420 data 0,31,128,0,0,0,0,0,0,0,0,0,0,0
1430 restore
```

A két példából látható, hogy bármelyik természettudományos tantárgyban könnyen található egyszerű játéklehetőség valamilyen élőlény, részecske mozgatásával, valamilyen mérhető mennyiség meghatározásával , stb. kapcsolatban. A humán tantárgyak területén talán még nagyobb szükség lenne ilyen szaktárgyi játékokra, mivel e tárgyaktól érthetően távolabb áll a számítógép.

10.2. Programvédelem

A játékprogramokat szokták leggyakrabban engedély nélkül másolni, ezért kívánkozik ide néhány gondolat a programvédelemről. A kereskedelemben vásárolható kész programok mindegyike komoly módszerekkel lett védve. E programok lemásolása gyakorlott programozóknak is nehéz feladat: a szerzők arra törekednek, hogy a védelem megfejtése olyan nagy munka legyen, amely már nem éri meg. Saját célra készített programnál miért lenne szükség védelemre? Elsősorban azért, hogy a valamennyit hozzáértő tanuló ne tudja megnézni a megoldásokat, ne tudja lemásolni, esetleg elrontani. Erre a célra igen egyszerű módszerek is megfelelnek, hiszen egy számítógéphez kitűnően értő tanuló sem "babrálhat" sokáig a programmal, úgy, hogy a tanár ne vegye észre. Gyakorlatilag a listázás, vagy a program megállításának letiltása már elegendő esetünkben. A továbbiakban ezekre a tiltásokra adunk ötleteket.

A listázást nehezen használhatóvá tehetjük a sorszámkiíratás tiltásával (C64 és C16, Plus/4):

POKE 22,35

engedélyezés:

POKE 22,25

A listázást teljesen megtiltja:

POKE 775,191 : C64 (engedélyezi: POKE 775,167)
POKE 774,187 : C16, Plus/4 (eng: POKE 774,110)
POKE 16863,195,204,6 : PRIMO

A listázás letiltásával a programok még megállíthatók és másolhatók. A SAVE parancsot is lehet hatástalanítani:

POKE 819,246 : C64 (eng: POKE 819,245)
POKE 816,136 : C16, Plus/4 (eng: POKE 816,164)

Az eddigieket a program elejére beírva a program elindításával fejtik ki hatásukat, de parancsként is használhatók. A C64-en a RUN/STOP és a RESTORE billentyűket, a PRIMO gépeken a RESET-gombot megnyomva a program leáll és a gép alapállapotba kerül: a programba írt fenti utasítások hatása elvész. Ezért ezeket is le kell tiltani. A STOP tiltása:

POKE 808,239 : C64
POKE 806,103 : C16, Plus/4

Visszakapcsolás:

POKE 808,237 : C64
POKE 806,101 : C16, Plus/4

A C64-en a RUN/STOP és a RESTORE együttes tiltását érjük el az alábbi utasítással:

POKE 808,254 , engedélyezés: POKE 808,237

Ebben az esetben nem lehet a programot megállítani, s ha az utolsó programsor GOTO-val az elsőre küld vissza, akkor kemény dió belenézni a programba.
A PRIMO RESET gombját az NMI kikapcsolásával lehet letiltani:

POKE 16443,PEEK(16443)AND 127:OUT 0,PEEK(16443)

Engedélyezés:

POKE16443,PEEK(16443) OR 128: OUT 0,PEEK(16443)

A C16 és Plus/4 gépek RESET gombját nem lehet ilyen egyszerűen letiltani, de nem is olyan fontos. E gépek csak a STOP és RESET egyidejű lenyomásával kerülnek a másik két géphez hasonlóan alapállapotba. A STOP gombot letiltva a RESET úgy szakítja meg a programot, hogy a BASIC-mutatókat is elállítja, tehát "amatőr" számára elvész a program: RUN, vagy LIST parancs nem találja a programot.

Nem interaktív programoknál (pl. szimuláció) letilthatjuk a teljes billentyűzetet, ekkor bármit csinálunk, fut a program:

POKE 649,0 : C64
POKE 1343,0 : C16 és Plus/4
POKE 16408,195 : PRIMO

A C64 népszerű segédprogramja a Simon's Basic is ad lehetőséget többféle védelemre, az alábbi könyv több gépre átvihető ötleteket mutat:

H.T.WELTNER: Tippek és trükkök a C64-hez. Data Becker-
-Novotrade, Budapest 1986.

A programvédelemmel kapcsolatban hívjuk fel a figyelmet arra, hogy a kereskedelemben készen kaphatók olyan gyorstöltő programok, melyek a betöltött programot azonnal indítják, nem adva lehetőséget a programban lévő védelem futtatás előtti kijátszására. Az automatikus indítás másik lehetősége, hogy betöltőprogramot készítünk, a programból töltött új program automatikusan indul:

```
10 rem"Betöltőprogram"
20 load"név"
```

Ha kazettáról töltünk, előtte a magnó PLAY gombja lenyomva kell legyen!

Befejezésül egy lényeges tanács: mindig legyen egy nem védett "törzspéldányunk" a programokból, nehogy a védelemmel öngólt rúgjunk!

11. Egyéb iskolai felhasználások

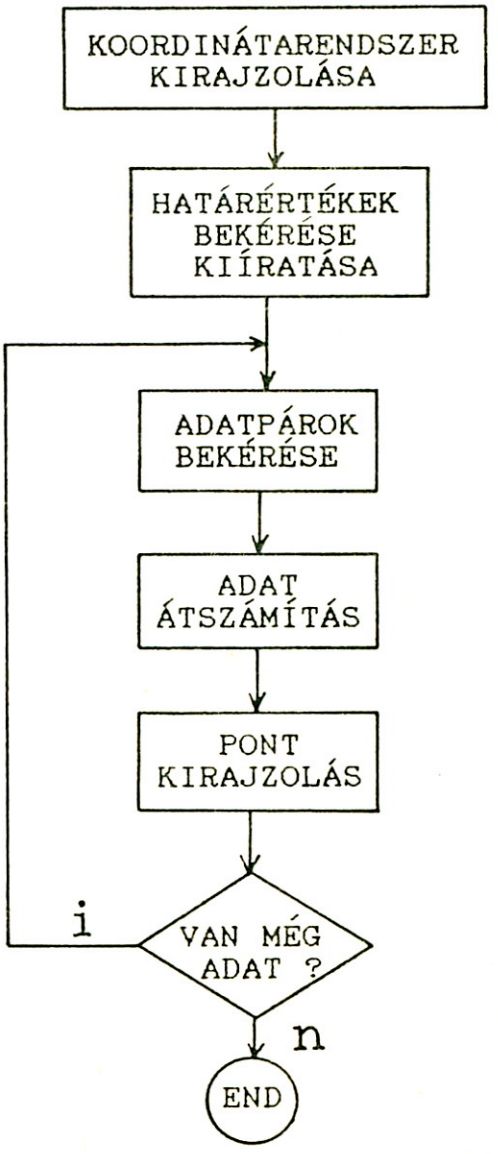
11.1. Adatfeldolgozás

A mai iskolákban rengeteg az olyan nem oktatási feladat, amely számítógéppel segíthető lenne. A pedagógusokra nagy terhet rónak az adminisztrációs feladatok: statisztikák készítése, felmérések értékelése, jelentések és beszámolók írása, leltározási munkák, könyvelések, órarend készítés, helyettesítések nyilvántartása, stb. Mindezek és még sok egyéb feladat megoldható számítógéppel. Egy viszonylag nagy iskola teljes adminisztrációját képes lenne ellátni a megfelelő perifériákkal rendelkező mikroszámítógép egy iskolai adminisztrátor kezelésében - akinek nem kell értenie a programíráshoz. Léteznek ilyen célú programok, de egy központi szerv könnyen készíttethetne minden iskolai feladatot ellátó programcsomagot. Iskolánként mindössze egy lemezmeghajtó és egy nyomtató szükséges a meglévő mikroszámítógép mellé - 1987-ben sok iskola vásárolhatott mindössze 15-20 ezer forintos áron ilyen perifériákat.

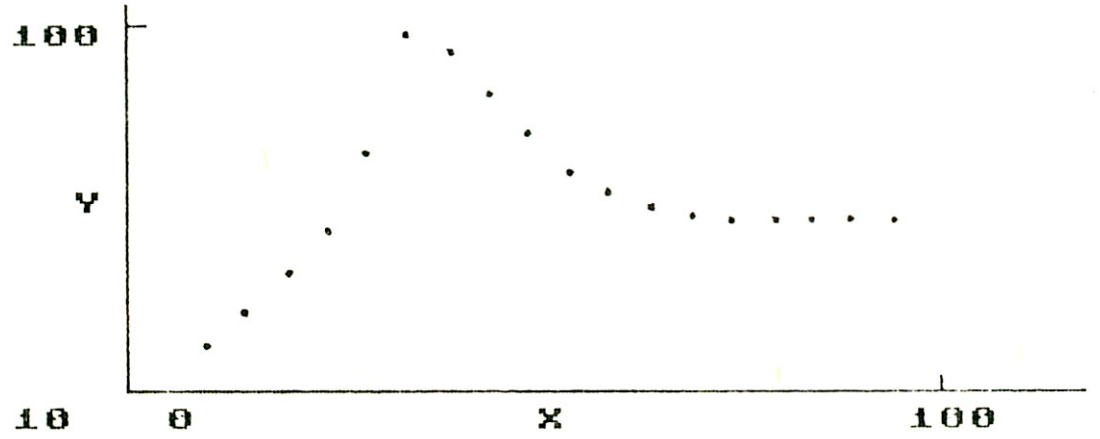
Néhány egyszerű, de a tanítás szempontjából is hasznos lehetőséget szeretnénk ebben a fejezetben bemutatni.

A mikroszámítógépek jó grafikai képességeit kihasználhatjuk különböző grafikonok készítésére, sőt egy ilyen programot megfogalmazhatunk általánosan: fizikai, kémiai, technikai tananyagokban, vagy a pedagógiai statisztikában összetartozó értékpárok ábrázolásával törvények felismerését könnyíthetjük anélkül, hogy a konkrét tananyaghoz kellene ezért programot írni. Egy nagyon egyszerű, de a fenti célra alkalmas kis program folyamatábrája, és a C16 vagy Plus/4 gépen futó listája:

```
10 rem grafikon
20 graphic2,1
30 draw,70,130 to 70,1
40 draw1,70, 8 to 75, 8
50 draw,70,130 to 318,130
60 draw1,281,130 to 281,125
70 char,7,8,"y"
80 char,22,17,"x"
90 input"legkisebb y-ertek:";y1
100 char1,4,17,str$(y1)
110 input"legnagyobb y-ertek:";y2
120 char1,4,1,str$(y2)
130 input"legkisebb x-ertek:";x1
140 char1,9,17,str$(x1)
150 input"legnagyobb x-ertek:";x2
160 char1,33,17,str$(x2)
170 xl=210/(x2-x1)
180 yl=120/(y2-y1)
190 print"ertekparok beadasa:"
200 input"x=";x
210 input"y=";y
220 if x=0 and y=0 then graphic0:end
230 if x1<1 thenx =x +1
240 if y1<1 theny =y +1
250 draw,(x-1)*xl+70,130-(y-1)*yl
260 goto 200
```



Grafikus képernyőre a CHAR utasítással csak sztringeket lehet kiíratni, változó aktuális értékét közvetlenül nem, ezért szükséges egy kis trükk: X1, X2, Y1, Y2 értékét az STR$ függvény sztringként állítja elő, így már ki lehet írni.
A 170–180-as sorok állítják elő azt az arányossági tényezőt (XL, YL), amely a kirajzolt koordinátarendszer és a beadott szélső értékek közti viszonyt fejezi ki. Az ábrázolandó pozíciókat a beadott adatpárok ezekkel való szorzásával kaphatjuk meg (250 -es sor). A 230–240-es sorok arra a gyakori esetre alkalmazzák a programot, amikor az ábrázolandó legkisebb érték 1-nél kisebb (többnyire 0).
A 200–260-as sorok közötti ciklus folyamatosan végzi az adatbekérést, átszámolást és rajzolást, a ciklusból kilépni – azaz a programot befejezni – 0,0 értékpár beadásával lehet (220-as sor). A következő ábra egy lehetséges képernyőt mutat:



A tudományok fejlődésének egyik fokmérője a kvantitatív módszerek alkalmazása. A pedagógiában néhány évtizede kezdődött a matematikai statisztika intenzív felhasználása, a széles körű, összehasonlító eredményvizsgálatok. Az eredménymérés elsődleges feladata a visszacsatolás a pedagógus számára, aki ez alapján határozhatja meg további stratégiáját – így valósul meg az oktatási folyamat irányítása. Az eredménymérés másik fontos feladata lehet hipotézisek értékelése, a hipotézisek igazolására szervezett kísérletek eredményeinek egzakt vizsgálata, különböző módszerek hatékonyságának megállapítása. Az ilyen pedagógiai vizsgálatoknak sok buktatója van: a minta kiválasztása, a mérés objektivitása, az adatok csoportosítása és feldolgozása. A buktatók elkerülését bőséges szakirodalom segíti [22,23,24], a továbbiakban két rövid programmal ötletet adunk a mikroszámítógépek bevonására. A pedagógiában megszokott átlag (számtani közép) sokszor a lényeget fedi el, ezért szükséges a szórás vizsgálata is. Két eredménysor közötti különbség sokszor vezet egy tanítási módszer túlértékelésére, amit egy t-próba elvégzésével megelőzhetünk.

A számtani közép, vagyis az átlag kiszámítását minden pedagógus jól ismeri: az adatokat összeadjuk és elosztjuk az adatok számával, precízen:

$$m = \frac{\sum_{i=1}^{n} x_i}{n}$$

Az átlag nem ad információt az adatok közötti eltérésekről, amely a pedagógiában igen fontos. 10 db. jeles és 10 db. elégtelen osztályzat átlaga ugyanaz, mint 20 db. közepesé. Az adatok szóródását több számszerű jellemzővel vizsgálhatjuk. A standard eltérés (S) – amit többnyire egyszerűen szórásnak neveznek – azt jelenti, hogy mekkora az egyes

adatok átlagos eltérése az adatok átlagától, míg a relatív szórás (V) ezt az eltérést az átlag százalékában fejezi ki, ezért alkalmasabb az összehasonlításokra:

$$S = \sqrt{\frac{\sum_{i=1}^{n} (x_i - m)^2}{n}} \qquad V = \frac{S \cdot 100}{m}$$

Az átlaggal és a szórással egy-egy adatsort lehet jellemezni. A pedagógiai gyakorlat sokszor igényli két adatsor kapcsolatának vizsgálatát. A korrelációszámítás célja két adatsor között fennálló összefüggés szorosságának megállapítása. Egy valódi példa: a természettudományok egybehangolt oktatási kísérlete során megvizsgáltuk az egyes természettudományos tárgyak eredményei közötti korrelációt az egybehangolt tananyagot tanuló és a többi osztálynál egyaránt [25]. A korrelációs együttható (r) értéke utal a kapcsolat szorosságára: r=0.75 felett szoros, r=0.6–0.75 között közepes, r<0.5 esetén laza az összefüggés (r értéke 0–1 között lehet). A korrelációs együttható kiszámítása:

$$r = \frac{\sum_{i=1}^{n} d_{x_i} \cdot d_{y_i}}{n \cdot S_x \cdot S_y}, \text{ ahol } \quad d_{x_i} = x_i - m_x, \quad d_{y_i} = y_i - m_y$$

A szignifikanciavizsgálat két eredmény közötti különbség okát kutatja. A pedagógiai jelenségekben nagy szerepe van a véletlennek, ezért pl. két osztály teljesítményében mutatkozó különbség nem biztos, hogy lényeges, tartalmi különbséget jelent. E kérdés eldöntésében segít a szignifikanciavizsgálat. Itt nem mehetünk részletekbe, mindössze az egyik leggyakrabban használt számítási módra közlünk mintaprogramot. A kétmintás t-próba eldönti, hogy a matematikai statisztika szerint két eredménysor közötti különbség szignifikáns (lényeges, valódi), vagy csak véletlen tényezőkre vezethető vissza biztonsággal. Az alábbi képlettel kiszámított t-értéket össze kell hasonlítani a megfelelő táblázat [24] adatával, amennyiben annál nagyobb, úgy a két adatsor között szignifikáns a különbség:

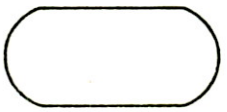
$$t = \frac{m_x - m_y}{\sqrt{\frac{\sum_{i=1}^{n} (x_i - m_x)^2 + \sum_{y=1}^{m} (y_i - m_y)^2}{n+m-2} \cdot \frac{n+m}{n \cdot m}}}$$

A fenti képletekben több helyen az adatok összegzését kell elvégezni. Az ilyen képletekkel való számolás sok időt igényel, pl. a kétmintás t-próba elvégzése egy harmincas létszámú osztály esetén órákat vehet igénybe. A programozható zsebszámológépek legjobbjai (pl.TI 58, 59) bizonyos ilyen statisztikai számításokra tartalmaznak kész programokat. Mikroszámítógéppel azonban igen gyorsan és kényelmesen végezhetjük el az ilyen vizsgálatokat. A beírt adatok a képernyőn ellenőrizhetők (RETURN előtt), az eredmény néhány másodperc alatt készen van. Az ilyen számítások programjaiban a lénye-

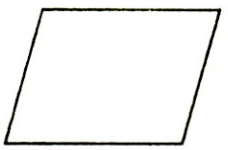
ges különbség az eddig tárgyaltakhoz képest, hogy a programnak tárolni kell és számítás közben többször újra felhasználni az adatokat. E műveleteket egyszerű ciklusokkal könnyen elvégezhetjük.

A kedves olvasó most ért el ahhoz a ponthoz, ahonnan kezdve egy hivatásos programozó is programnak nevezné a leírt listákat. Ebben van egy kis túlzás, de az kétségtelen, hogy az eddigi programok – és módszereik – csak oktatásra vagy játékra alkalmasak. A gyakorlati életben alkalmazott számítástechnika (adatfeldolgozás, nyilvántartás, tervezés, stb.) a továbbiakban látható kis programokhoz hasonló elemekből épitkezik.

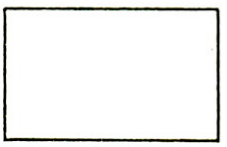
Az eddig látott folyamatábrák inkább a tanítás szempontjából próbálták bemutatni a feladat megoldását, ezért jelöléseik nem a számítástechnikában elfogadottaknak megfelelőek voltak. Most célszerűnek látszik megismerkedni néhány szabványos jelöléssel:



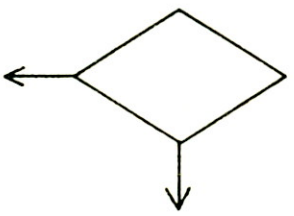
Határoló blokk (stop, vagy start)



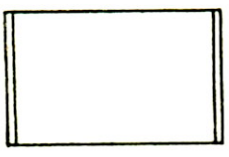
Beviteli, illetve kiviteli blokk
Itt a program adatot vár, vagy ad.



Általános műveleti blokk
A BASIC-ben lehetséges műveleteket tartalmazhatja.

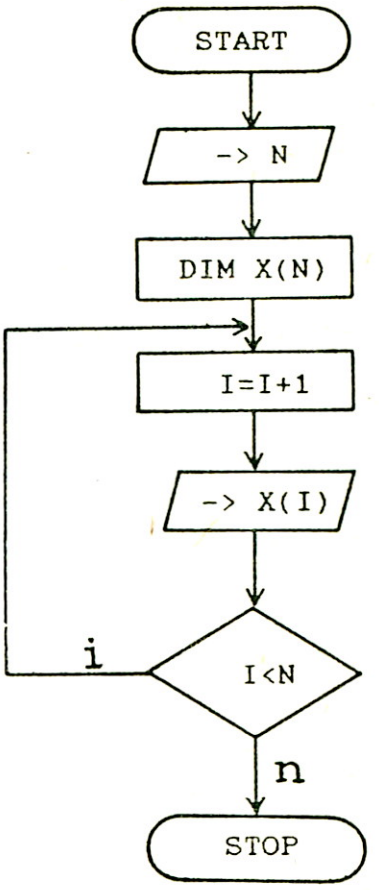


Elágazási, vagy döntési blokk
Egy feltételtől függően két irányban folytatódhat a program.



Egy máshol már részletezett, egyetlen start, illetve stop-blokkal határolt eljárás.

Az egyértelmű műveleti előírást, műveletsorozatot (pl.: vedd le a kagylót-dobj be két forintot-várd a búgó hangot...), amely valamilyen feladat megoldását eredményezi algoritmusnak szokás nevezni. A t-próba számítási algoritmusának megtervezése a probléma lényegének megoldását jelenti, a konkrét gépre való program megirása gyakorlott programozónak már csak rutinfeladat, de kezdőnek sem nehéz.



E kitérő után térjünk vissza az adatbevitel és tárolás megoldásához. Nagyobb mennyiségű adatot indexes változók értékeiként tömbökben lehet tárolni, előtte a tömböket dimenzionálni kell, azaz megfelelő tárhelyet biztositani. Az adatok beolvasása és a tömbökből való kiolvasása egyaránt ciklusokkal oldható meg. A jobb oldalon látható egy beolvasási ciklus folyamatábrája (N számú adatra):

A folyamatábra kódolása a megfelelő BASIC-program megirása:

```
10 INPUT N
20 DIM X(N)
30 I=0
40 I=I+1
50 INPUT X(I)
60 IF I<N THEN 40
70 END
```

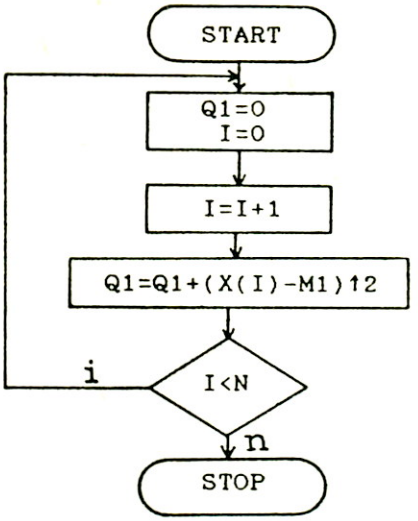
Ez a kis program N számú adatot beolvas és tárol. A ciklust másképpen is szervezhetjük, a FOR...NEXT utasitáspárral:

```
30 FOR I=1 TO N
40 INPUT X(I)
50 NEXT I
60 END
```

Ez utóbbi megoldás a megszokottabb, gyorsabb.
Az adatok összegzését tartalmazó képletek alapján való számitás is ciklussal oldható meg. A t-próba egy részképlete:

$$Q = \sum_{i=1}^{n} \left(X(i) - M1\right)^2$$

A kiszámítás folyamatábrája:



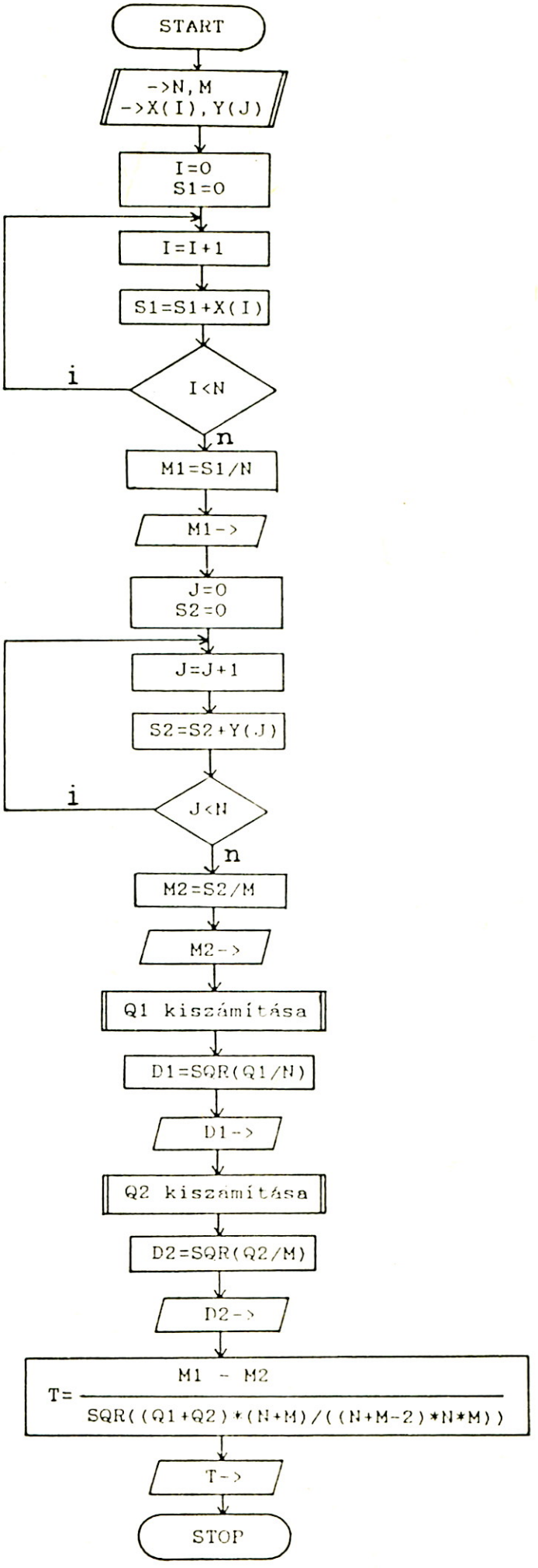
Természetesen ezt a folyamatrészt meg kell előzze az adatbevitel, amely egy tömbben letárolta az X(I) adatokat, ez a programmodul onnan olvas. A megfelelő lista:

```
100 Q1=0
110 FOR I=1 TO N
120 Q1=Q1+(X(I)-M1)!2
130 NEXT I
```

Ez a programrész semmi "láthatót" nem csinál, csak a meglévő adatokból és a már kiszámitott M1 (átlag) értékéből kiszámol valamit és ezt Q1 változó értékeként tárolja a további felhasználásig. Amennyiben érdekes Q1 értéke, természetesen egy PRINT utasitással ki is lehet iratni.

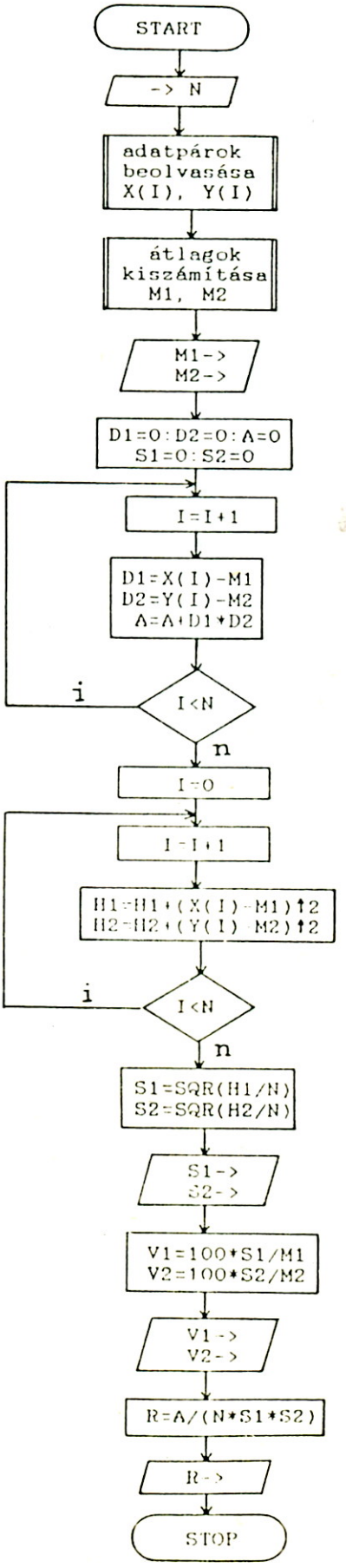
A következő oldalon látható a kétmintás t-próba kiszámításának folyamatábrája, és a bármely gépen működő BASIC-programja.

```
100 rem "t-proba"
110 input"adatok szama 1.=";n
120 input"adatok szama 2.=";m
130 dim x(n),y(m)
140 print"1. adatsor"
150 for i=1 to n
160 input x(i)
170 next i
180 print"2.adatsor"
190 for j=1 to m
200 input y(j)
210 next j
220 s1=0
230 for i=1 to n
240 s1=s1+x(i)
250 next i
260 m1=s1/n
270 print"1. atlag=";m1
280 s2=0
290 for j=1 to m
300 s2=s2+y(j)
310 next j
320 m2=s2/m
330 print"2. atlag=";m2
340 q1=0
350 for i=1 to n
360 q1=q1+(x(i)-m1)↑2
370 next i
380 d1=sqr(q1/n)
390 print"1. standard elteres=";d1
400 q2=0
410 for j=1 to m
420 q2=q2+(y(j)-m2)↑2
430 next j
440 d2=sqr(q2/m)
450 print"2. standard elteres=";d2
460 t=(m1-m2)/sqr((q1+q2)*(n+m)/
((n+m-2)*n*m))
470 print "t=";t
480 end
```



A korrelációs együttható kiszámítása hasonló programmal lehetséges. A látott képlet alapján való számítás folyamatábrája, és a BASIC-program (bármely gépre):

```
10 rem korrelacio
20 print"S"
30 input"adatparok szama:";n
40 dimx(n),y(n)
50 print"adatparok:"
60 for i = 1 to n
70 input x(i),y(i)
80 next i
90 p1=0: p2=0
100 for i=1 to n
110 p1=p1+x(i)
120 p2=p2+y(i)
130 next i
140 m1=p1/n: m2=p2/n
150 print"atlagok:"
160 print"m1=";m1,"m2=";m2
170 d1=0:d2=0:a=0:s1=0:s2=0
180 for i=1 to n
190 d1=(x(i)-m1)
200 d2=(y(i)-m2)
210 a=a+d1*d2
220 next i
230 h1=0:h2=0
240 for i=1 to n
250 h1=h1+(x(i)-m1)↑2
260 h2=h2+(y(i)-m2)↑2
270 next i
280 s1=sqr(h1/n)
290 s2=sqr(h2/n)
300 print"standard elteresek:"
310 print"s1=";s1,"s2=";s2
320 print"relativ szorasok:"
330 print"v1=";100*s1/m1;"v2=";100*s2/m2
340 r=a/(n*s1*s2)
350 print"korrelacios koefficiens=";r
360 end
```



Észrevehetjük, hogy az ilyen programok mennyire kevéssé gépfüggőek. A programirás lényege a megoldás algoritmusának elkészítése, a folyamatábra (műveletterv) összeállítása. Ennek ismeretében már kevés gyakorlattal bármilyen gépre megírhatjuk a programot, szerencsés esetben – mint fent – csak a legáltalánosabb (minden gépen azonos) utasításokra van szükségünk.

Több iskolai adminisztrációs feladatra lehet kész programokat kapni, ilyen az órarendkészítés is. Elképzelhető, hogy ideális esetben (kevés osztály, sok tanár, sok terem) ezekkel a programokkal lehet jó órarendet készíteni. A valóság azonban sok nehezen paraméterezhető kívánságot támaszt az órarendekkel szemben: személyes kívánságok, szaktantermek, kisérletes tárgyak igényei, stb. Általános tapasztalatok szerint még sokáig megmarad az órarend hagyományos készítése...

A pedagógiai – statisztikai vizsgálatokat segitő programok készítéséről mondottakat összefoglalva egy javasolt munkaszervezési sorrend:

- Alapos szakirodalmi tájékozódás a statisztikai mutatók pontos értelmezéséről.
- A vizsgálat pontos céljának és majdani következtetéseink korlátainak megállapitása.
- A számitási képletek kisebb részekre bontása (bonyolultabb képleteknél).
- Az egyes rész-számitások folyamatábráinak kidolgozása.
- A rész-számitások eredményeinek képlet szerinti összegzéséhez szükséges folyamatábra (főprogram) kidolgozása.
- Az adatbevitel és tárolás folyamatának megszervezése (folyamatábra).
- Az adatok kiiratásának megszervezése.
- A program megirása.
- Kipróbálás ismert adatok (szakirodalom), vagy egyszerű számokból kézzel kiszámolt eredmények alapján.

11.2. Perifériák

Perifériáknak szokás nevezni a számítógépek központi egységéhez csatolható különböző készülékeket, melyek megléte nem feltétele a számítógép működésének. A mikroszámítógépek leggyakoribb perifériái:

- kazettás egység (magnó)
- mágneslemez-meghajtó (floppy)
- nyomtató
- botkormány (joystick), fényceruza

A Commodore gépeket rendszerint speciális kazettás egységükkel együtt forgalmazzák, míg a PRIMO, TVC és a legtöbb mikrogép kommersz magnetofont használ kazettás egységként. E magnók olcsó, egyszerű kazettákon tárolnak adatokat, programokat, de csak sorosan, ezért használatuk kényelmetlen. Legjobb kizárólag rövid játékidejű kazettát használni és minden programot külön kazettán tárolni - így a keresgélés kiküszöbölésével csökkenthetjük a magnók lassúsága miatti időveszteséget. Feltétlenül indokolt többféle gyorstöltő programot (turbo) is beszerezni a kereskedelemtől, vagy gyakorlottabb programozóktól. Ezek a programok 10-15-szörösére sűritik az adatokat, így ennyiszer gyorsabb a betöltés és kimentés, ennyivel kevesebb szalag szükséges. Néhány turbo-program cime: HER-turbo, Turbo-16, Turbo-64, Turbo Typ.

A mágneslemez-meghajtó (floppy disk drive), vagy floppy lényegesen nagyobb hatásfokú adat- és programtárolást valósit meg, sajnos az ára az alapgépek árának 2-3-szorosa. Ezek a készülékek kisméretű hajlékony mágneslemezt használnak tárolásra, egy lemez kapacitása kb. 170 Kbyte. A lemezmeghajtóknak saját mikroprocesszoruk és operációs rendszerük van, képesek bizonyos tevékenységeket a számítógép utasitásai után önállóan végezni, miközben a számítógép más feladatot hajt végre.

Hazánkban legjobban elterjedt a Commodore cég VIC 1540/1541 tipusú készüléke, mely minden Commodore géphez - megfelelő interface segitségével egyéb gépekhez is (pl. PRIMO) - csatlakoztatható. Floppy segitségével megoldhatjuk C64 programok C16, vagy Plus/4 gépbe töltését, ami az eltérő kazettás egységekkel közvetlenül nem lehetséges. Mivel a C16 és Plus/4 felülről kompatibilis a C64-el, ezért minden olyan C64-re irt BASIC-program, amelyik nem tartalmaz gépi kódú részeket, POKE és PEEK utasitásokat, futtatható a C16-on és Plus/4-en. C64-programok fejlesztéséhez is igénybevehetünk Plus/4, illetve C16 gépet olyan műveletekhez, amelyeket a C64 nem tud,pl. átsorszámozás, DELETE parancs. A C16-nál azonban a jóval kisebb tárkapacitás akadályt jelenthet.

A nyomtató a képernyőhöz hasonló output (kimeneti) eszköz, de a vezérlő karakterek értelme más, mint a képernyőn. A nagyobb gépekhez egyszerre egész sorokat nyomtató igen gyors sornyomtatókat használnak, míg a mikrogépekhez szinte kizárólag mátrixnyomtatókat. Az elnevezés arra utal, hogy a képernyőhöz hasonlóan pl. 8x8-as, vagy egyéb méretű pontrácsból áll össze egy kinyomtatott karakter. A papir előtt gyorsan mozgó nyomtatófejben egymás felett 7-9 tű van, melyek impulzust kapva rácsapnak a festékszalagra, ami a papirhoz nyomódva pontot hagy. A legtöbb ilyen nyomtató másodpercenként 20-100 karaktert képes kinyomtatni, a gyorsabbak nemcsak balról jobbra tudnak irni, hanem visszafelé is, igy nincs üresjáratuk.

A nyomtatók saját ROM-jukban tárolják a nyomtatható karakterek képeit, a Commodore nyomtatói (MPS 801,MPS 802) az alapgépről ismert grafikus karaktereket is. A modernebbek (pl. MPS 1000) többféle nemzeti karakterkészletet is tárolnak, és a számítógép segitségével szerkeszthetünk tetszőleges új karaktereket. A nyomtatók egy része rendelkezik pontgrafikus üzemmóddal, amelyben a grafikánál látottakhoz hasonlóan rajzok

készithetők. Sajnos még a Commodore nyomtatói sem teljesen kompatibilisek, röviden a legismertebbek:

MPS 801 és 803 :karakteres és grafikus üzemmód, sorsűrűség állandó.
MPS 802 : csak karakteres üzemmód, sorsűrűség programozható.
MPS 1000: előző kettővel felülről kompatibilis, nemzeti karakterkészletek, IBM gépekkel is kapcsolható.

A nyomtatók elsőrendű felhasználása a programlisták kinyomtatása. Aki irt már hosszabb programot, az tudja, milyen nehéz programirás közben a hosszú listában a képernyőn kikeresni bizonyos elágazásokat, értékadásokat, stb-t. A nyomtatóknak azonban ennél sokkal nagyobb a jelentőségük, elsősorban két területen: rajzeszközként és irógépként.

A grafikus üzemmóddal rendelkező nyomtatók képesek a képernyő tartalmának papirra irására. Erre a célra egy külön program, úgynevezett 'hard copy' szükséges. Az eredeti programmal - vagy parancs módban - elkészitett rajz után GRAPHIC 0 (C16, Plus/4) utasitással karakteres üzemmódba váltunk, NEW paranccsal töröljük a programot (a törlés nem minden hard copy programnál kötelező) és betöltjük a 'hard copy' programot. Ezután a megadott módon elindítva papirra másolja a - képernyőn nem is látszó - grafikus képernyőtartalmat. Megoldható a programba épitett rajzoló szubrutin is, amelyik felhasználói kivánságra kinyomtatja az aktuális grafikus képernyőtartalmat. Egy ilyen program listája megtalálható Ury L.: Commodore C-16, C-116 című könyvének (LSI ATSZ kiadás) 236. oldalán. A C64 népszerű segédprogramja, a Simon's Basic rendelkezik nyomtató parancsokkal:

COPY : a grafikus képernyő-tartalmat nyomtatja
HRDCPY : a karakteres képernyő-tartalmat nyomtatja

A tanitásban kitűnően lehet hasznositani a nyomtatókat. A tanulók munkája végeztével, vagy akár közben irásbeli értékelést adhatunk, jutalomként rajzot, diszes oklevelet, stb. Az előző fejezetben leirt grafikon-programhoz is hozzákapcsolható a kész grafikont kinyomtató program.

A nyomtatók irógépkénti alkalmazása napjainkban egyre jobban terjed. Sokféle szövegszerkesztő program van forgalomban, a Plus/4 alapkiépitésben tartalmaz ilyen programot. A jobb szövegszerkesztő programok többféle nyomtatót tudnak kezelni, sőt megfelelő interface közbeiktatásával IBM, vagy ROBOTRON villanyirógépet is. Szinte minden mikroszámitógéphez kaphatók szövegszerkesztő programok kazettán, vagy lemezen. A legelterjedtebb hazánkban a C64-en futó 'EASY SCRIPT' nevű program, melynek több 'magyaritott' - azaz ékezetes betűket is iró - változata ismeretes.

A szövegszerkesztő programok szabad formátumos bevitelt biztositanak, azaz gépeléskor nem kell semmire figyelni, csak a szövegre. A sorok hosszát, sorközt, lapszámozást, jobbra, balra igazitást, stb. formátum parancsokkal állithatjuk be gépelés előtt, vagy után. A képernyőn a kiviteli formátumnak megfelelő alakban is végignézhetjük a szöveget nyomtatás előtt. Bármikor lehet a szövegben javitani, szavakat, mondatokat, bekezdéseket áthelyezni, törölni, beszúrni. Az elkészitett iratok lemezen, vagy kazettán tárolhatók, igy később újrafelhasználhatók módosithatók. Az EASY SCRIPT postai szerkesztésre is alkalmas: megadott listán szereplő cimekre megcimezi az iratokat, körleveleket, kitölt űrlapokat, stb. A kommunális ellátást végző vállalatok rendszerint nagyszámitógépekkel készitik a számlákat, de ugyanezt - kevesebb adattal - egy mikroszámitógépes szövegszerkesztővel is elvégezhetjük.

Összefoglalva elmondhatjuk, hogy egy titkárnő, vagy sokat gépelő, levelező szakember szövegszerkesztőt megismerve többet nem akar nélküle dolgozni.

12. Irodalomjegyzék

12.1 Hivatkozások

1. B.F.Skinner: The science of learning and the art of teaching. Harvard Educational Review,1954.

2. G. Leith: Second thoughts of programmed learning. London, 1969.

3. N.A.Crowder: Automatic teaching.Wiley,New York,1959.

4. Deák L,Gyülvészi P.: Módszertani Közlemények, XIV.évf. 278–280.p. 1974.

5. Deák L.: Módszertani Közlemények XV.évf. 34–38.p. 1975.

6. Deák L.: A tanulmányi teljesítmény ellenőrzése a magyar felsőoktatásban. Tanulmánykötet, FPK Budapest, 1979. 304–334.p.

7. Kiss Á.: A tanulás programozása. Tankönyvkidó, Budapest, 1973

8. Sárik T.: A Kémia Tanítása. 1971. 33.p.

9. Garami K.: A programozott oktatás. Eredmények és feladatok. OPI, Budapest, 1969. 247.p.

10. Michalovszky Cs.–né: Programozás a kémiában. Tankönyvkiadó, Budapest, 1971.

11. J.B.Pattison: Programozott bevezetés a gáz–folyadék kromatográfiába. Műszaki Könyvkiadó, Budapest, 1975.

12. Szabó S.–né: Pécsi Tanárképző Főiskola Tudományos Közleményei. Tom.18.Ser.6. 1974.

13. Szebeni Sz.,Müllner E.: Szakmódszertani Közlemények ELTE TTK VIII. 95.p. 1975.

14. Fürjes J.,Biszterszky E.: Tanítógépek és programok. OMKDK, Budapest, 1972.

15. W.R.Fuchs: Az új tanulási módszerek. Közgazdasági és Jogi Könyvkiadó, Budapest,1971.

16. Vári P.: A programozott oktatás új irányzatainak kritikai elemzése. OOK, Veszprém, 1982.

17. C.Kupisiewicz: Felsőoktatási Szemle 24. 481.p. 1975.

18. J.S.Bruner: Az oktatás folyamata. Tankönyvkiadó Budapest, 1968.

19. Deák L.,Dévényi I.-né,Síposné Kedves É.: Kémia 8. Tankönyvkiadó, Budapest, 1987.

20. Agócs L.: A Fizika Tanítása XXIV. 2. 60.p. 1986.

21. Fürjes J.-Biszterszky E.: Tanítógépek és programok. OMKDK Budapest, 1972.

22. Ágoston Gy.,Nagy J.,Orosz S.:Méréses módszerek a pedagógiában. Tankönyvkiadó, Budapest, 1979.

23. Nagy J.: A témazáró tudásszintmérés gyakorlati kérdései Tankönyvkiadó, Budapest, 1972.

24. Hajtman B.: Bevezetés a matematikai statisztikába. Akadémiai Kiadó, Budapest, 1968.

25. Síposné Kedves É.,Deák L.: Módszertani Közlemények XXVI.évf.3.sz. 174.p.

12.2 Ajánlott irodalom

12.2.1. Könyvek

Felhasználói kézikönyvek: C64, C16, plus/4, PRIMO, TV-Computer

Hámori M.: Tanulás és tanítás számítógéppel. Tankönyvkiadó, Budapest, 1984.

Bodor T.,Gerő P.: A BASIC programozás technikája. SZÁMALK, Budapest, 1983.

Mágoriné Huhn Á.,Puskás A.: Számítógép az általános iskolában. Módszertani Közlemények Könyvtára, Szeged, 1986.

Hetedhét... sorozat. NOVOTRADE, Budapest 1985-87.

Úry L.: Commodore 64. I-II. LSI ATSZ, Budapest, 1985.

Angerhausen,Englisch,Gerits: Tippek és trükkök a Commodore 64-eshez. Data Becker - Novotrade, Budapest, 1985.

Weltner H.: További tippek és trükkök a Commodore 64-eshez. Data Becker - Novotrade, Budapest 1986.

Plenge,Szczepanowsky: Simon's Basic gyakorlatok. Data Becker - Novotrade, Budapest 1986.

Liesert: PEEK-ek és POKE-ok a C 64-esen.Data Becker - Novotrade, Budapest, 1986.
Angerhausen,Brückman,Englisch,Gerits: A Commodore 64-es belső felépitése. Data Becker - Novotrade, Budapest, 1985.

Kőhegyi J.(szerk): Ismerd meg a BASIC nyelvjárásait! sorozat, Műszaki Könyvkiadó, Budapest 1986.

Úry L.: Commodore C-16, C-116. LSI ATSZ, Budapest, 1986.

Theisz Gy.: BASIC tanácsadó C-16, plus/4. Műszaki Könyvkiadó, Budapest, 1987.

Tóth V.: A Commodore 16-os belső felépitése. Novotrade, Budapest, 1986.

Babán G.,Masa I.: Gépi kódú programozás kezdőknek és haladóknak. Novotrade, Budapest, 1987.

PRIMO füzetek, szoftver. MTA-SZTAKI-COSY, Budapest, 1985.

PRIMO füzetek, hardver. MTA-SZTAKI-COSY, Budapest, 1985.

Soós G.,Szilassi L.: Algoritmusok, játékok. OPI, Budapest, 1988.

Bencsikné Takács M.: Feladatgyüjtemény C16-os számitógéphez általános iskolásoknak. Novotrade, Budapest, 1986.

Lőcs Gy.: A BASIC és a Kiváncsi. Tankönyvkiadó, Budapest, 1986.

Lőcs Gy.: A BASIC és a kiváncsi Feladatgyüjtemény. Tankönyvkiadó, Budapest, 1986.

Csákány A.,Vajda F.: Játékok számitógéppel. Műszaki Könyvkiadó, Budapest, 1980.
Dedinszky F.,Horányi I.: Számitástechnika a történelem tanitasában. Novotrade, 1987. ("Suiikomp" sorozatban)

12.2.2. Folyóiratcikkek

Solti A.: Ötlet 6. évf.18.sz. 30-35.p. (1987. ápr.30.) (életjáték C64-re)

Horváth Z.: Ötlet 6. évf. 28.sz. 13.p. (1987.júl.9.) (szintetizátor C64-re)

Hábeller Zs.: Ötlet 6. évf. 40.sz. 30.p. (1987.okt.1.) (hang C16-ra)

Ötlet 6. évf. 35.sz. 39.p. (1987.aug.21.) (ABC szerinti rendezés)

Fekete Gy.: Ötlet 6. évf. 40.sz. 28-30.p. (1987.okt.1.) (PRIMO rajzolás)

Szilassi L.: Ötlet 5.évf. 1986.jún.26. 36-37.p. (PRIMO térbeli rajzolás)

A mikroszámitógépek konkrét tanitási alkalmazásairól:

A FIZIKA TANÍTÁSA:

Kalamár Cs.,Papp Gy.-né: XXII.évf. 2.sz. 42-46.p. (1983) (statisztikus jelenségek)

Farkas O.-né: XXIV.évf. 6.sz. 170-174.p. (1985) (demonstrációs és gyakorló program PRIMO-ra)

Agócs L.: XXIV.évf. 2.sz.60-61.p. (1986) (gyakoroltató programok)

Nagy L.,Szegedi E.: XXV.évf.3.sz. 73-77.p. (1986) és 5.sz. 144-177.p. (feladatok egyéni tanuláshoz)

Gudenus L.-né:XXV.évf.3.sz.78-79.p.(1986) (fogyasztók kapcsolása)

Isza S.: XXV.évf.3.sz. 80-84.p. (1986) (mozgás kiértékelés)

Kósa J.: XXVI.évf. 3.sz. 93-94.p. (1987) (stopper nagy számjegyekkel)

Hevér M.,Szabó S.,Harangozó J.: XXVI.évf.1.sz. 12-16.p. (1987) (Millikan kisérlet szimulációja)

Takács E.: XXV. 1.sz.12-16.p. (1986) (mechanikai demonstrációs kisérletek)

Zátonyi S.: XXV. 1.sz. 16-21.p. (1986) (szimulációs, demonstrációs programok)

Türi L.: XXV. 4.sz. 119-121.p. (1986) (hangsebesség mérés vezérlése)

Szatmári R.: XXV. 4.sz.122.p. (1986) (jelek, mértékegységek gyakorlása)

Smidéliusz Zs.,Zátonyi S.:XXVI.évf.4.sz.125-127.p. (1987)

FIZIKAI SZEMLE

XXXII.évf. 7-8.sz. több cikk (1982) (programozható zsebszámológép)

A BIOLÓGIA TANÍTÁSA:

Majer J.,Bordács M.: XXVI.évf. 1.sz. 18-21.p. (1987) (munkafüzeti kérdések lineáris programja)

Demeter T.,Békés F.,Szathmáry E.: XXVI.évf. 6.sz. 172-177.p. (1987) (növények tápanyag-felvétele, grafikus szimulációs program)

Nagy L.,Takács T.: XXVI. évf. 6.sz. 177-179.p. (1987) (öröklődési program)

A KÉMIA TANÍTÁSA

Hobinka I.,Riedel M.: XXII. évf. 112.p. (1983)

Riedel M.,Hobinka I.: XXIII. évf. 102.p. (1984)

Deák L.: XXIII.évf. 5.sz. 141-145.p. (1984) (zsebszámológép programok)

XXV.évf.1.sz. 30-32.p. (1986)

Clugston M.: XXV.évf. 5.sz. 134-138.p. (1986) (nemzetközi tapasztalatok)

Zsidai E.: XXV.évf. 5.sz. 138-142.p.(1986)
XXVI.évf. 2.sz. 60-63.p. (1987)
XXVI.évf. 3.sz. 87-90.p. (1987) (zsebszámítógép programok)

Győri E.: XXVI.évf.2.sz. 63-65.p. (1987) (Novotrade programok ismertetése)

Berecz Á.-né,Deák L.,Nagyné Várkonyi K.:XXVII.évf. 2.sz. (1988) (kutatási terv és programok ismertetése)

Berecz Á.-né,Deák L.,Nagyné Várkonyi K.:XXVII.évf. 3.sz. (1988) (számítógép lehetőségei a kémiaoktatásban)

A TECHNIKA TANÍTÁSA

SZ.Lukács J.:XVI.évf. 6.sz. 169-175.p.(1984) (a számítógép lehetőségei az általános iskolai technika tanitásában)

Papp S.:XVII.évf.2.sz. 33-39.p.(1985) (javaslat számítás-technikai képzésre)

Illés L.-né:XVIII.évf.1.sz. 5-10.p.(1986) (számítástechnika külföldön)

Kőrösné Mikis M.:XVIII.évf.1.sz. 13-17.p. (1986) (számítógépek ergonómiai kérdései)

Kőrösné Mikis M.:XIX.évf.2.sz. 60-64.p.(1987) (új perspektivák a számítógépek iskolai alkalmazásában)

NYELVOKTATÁS

Tokody L.: Idegen Nyelvek Tanitása 1987. 6.sz. 185-188.p. (számítógép az angol tanitásában)

Papp F.: Magyar Nyelvőr 108.évf. 1.sz. 8389.p. (1984) (magyar nyelvtan oktatása)
Papp F.: Számítógép és nyelvoktatás. VEAB Értesitő, 1983. I., II.

TÖRTÉNELEM

Sávoly M.,Jancsó F.: Pedagógiai Szemle XXXIII.évf. 1.sz. és XXXIV. 11. sz. 1071-1079.p. (gyakorló feladatlapok feldolgozása)

RAJZ

Lázárné Berkecz É.: Rajztanitás XXVIII.évf. 1.sz. 3-7.p. (1986)

MEGRENDELŐ

Megrendelem az LSI ATSZ kiadásában megjelenő **'Dr. Deák László: Mikroszámitógépes oktatóprogramok készitése és alkalmazása'** című kiadványt ... példányban. (Ára:175.-Ft)
Továbbá oktató kazettát ..... géptípusra ... példányban. (Irányára 200-300 Ft között a kazetta méretétől függően.)
Fizetés utánvéttel vagy átutalással. Szállítás megjelenés után azonnal, vagy .....
.....

Megrendelő neve:

Szállítási cím:

Ügyintéző neve, telefonszáma:

LSI Alkalmazástechnikai Tanácsadó Szolgálat
Budapest 1300, Pf.: 114.

Ára:175,–Ft

VÉGRE!
EGY ÉRTELMES JÁTÉK,
amely igaz, hogy tanít,
de a játszás örömével együtt.

A panelen 24-féle elektronikus készülék alakítható ki. Élvezetesebbé teszik használatát a fényelemek. Lehet csupán játék, lehet betekintés azokba a tudományokba, amelyek ismerete a közeljövőben meghatározza életünket. Tartalma:

Játékok – ugyanakkor **valóságos készülékek**

– **Rádió vevők;** – **számítógép alapáramkörök;**
– **rádió adók;** – **elektronikus cica, csipogó, morse**

és mindezek nap- és fényenergiával működtetve vagy vezérelve.

Kapható:

2C Áruház, XIII. Balzac u. 35.
ALFACORD GT VIII. József krt. 40.
ALFADAT GT Tatabányai Centrum Áruház
Fókusz Könyváruház, VII. Rákóczi út 14.
FOTOELEKTRONIK GT
– vevőszolgálat: VII. Dohány u. 16
– szervizeiben vidéken is
MIGÉRT bemutatóterem, VIII. Rákóczi út 57.
MOZAIK Üzletház, V. Petőfi Sándor u. 10.
OTTHON Áruház – műszaki osztály
TECHNIKA Könyvesbolt, XI. Bartók B. út 15.
ÚTTÖRŐ Áruház – műszaki osztály

Megrendelhető illetve megtekinthető: LSI ATSz, 1033 Budapest III. Hévizi út 6/e.

Ára: 1944.– Ft