

**Cím: DIDAKTIKUS PASCAL**

**Szerző: Fekete A. Albert-Zsombor**

**Oktatási intézmény:** Bolyai Farkas Elméleti Líceum  
Marosvásárhely

# Pascal programozás didaktikusan

## Pascal programozás didaktikusan

### Tartalomjegyzék

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| 1. ...Bevezető – általánosítás .....                            | 3.  |
| 2. Program használati útmutatója – felhasználói kézikönyv ..... | 4.  |
| 2.1. Főmenü .....                                               | 4.  |
| 2.2. Gráfelmélet .....                                          | 5.  |
| 2.2.1. Grágépítő.....                                           | 5.  |
| 2.2.2. Szélességi bejárás.....                                  | 6.  |
| 2.2.3. Mélységi bejárás.....                                    | 6.  |
| 2.2.4. Összefüggő komponensek.....                              | 6.  |
| 2.2.5. Legrövidebb utak.....                                    | 6.  |
| 2.2.5.1.Prím.....                                               | 6.  |
| 2.2.5.2.Kruskal.....                                            | 6.  |
| 2.2.6. Euler vonal.....                                         | 7.  |
| 2.2.7. Hamilton kör.....                                        | 7.  |
| 2.3. Rendezés.....                                              | 7.  |
| 2.4. Memoriaterület.....                                        | 7.  |
| 2.4.1. Sor.....                                                 | 7.  |
| 2.4.2. Verem.....                                               | 7.  |
| 2.4.3. Lista.....                                               | 8.  |
| 3. Programozási háttér.....                                     | 8.  |
| 4. Következtetések.....                                         | 13. |
| 5. Függelékek.....                                              | 14. |

## 1. Bevezető - általánosítás

Napjainkban, a középiskolás oktatásban egyre nagyobb teret kapnak a számítógépes bemutatók. A vizuális formában megjelenített anyag sok diák számára, köztük számomra is, sokkal inkább elérhetőbbnek, könnyebben elsajátíthatobbnak minősül. Az érdekes és a táblára felrajzolt ábráknál jóval pontosabb, esztétikailag magasabb szintet, szépséget biztosító rajzok szemléletesebbé teszik a biológiaórákat, segítenek könnyebben hozzájutni a kiszabott információanyaghoz, továbbá a pontos grafikonok, mozgóképek (gif, flash, avi, stb), vizuális kísérletek látványosabbá varázsolják a fizika-, informatika- és kémiaórákat. Számos olyan eljárás, kísérleti szimuláció kerülhet ezúton bemutatásra, amire egyébként nem nyílik lehetőség egy középiskola szaklaboratóriumában.

Ez az elgondolás ihletet adott a dolgozatom témájának a rögzítésére. A IX-XI.-es informatika tananyag (**Pascal programozási nyelv**) vizuális környezetben való megjelenítése célszerű feladatnak bizonyult. Célkitűzésem a három tanév alatt elsajátított valamennyi jellegzetes ciklus (minimumkiválasztásos rendezés), programrészlet (listás adatstrukturák), működési koncepció (**LIFO**<sup>1</sup> – sorszimuláció, **FIFO**<sup>2</sup> – veremszimuláció, adatok lista segítségével történő rögzítése, valamint a velük elvégezhető műveletek), nagyobb fejezet (fraktálok, gráfelmélet) grafikus úton történő bemutatása a Pascal programozási nyelvben.

A programcsalád megszerkesztésének állapotában kiemelt figyelmet irányítottam arra a tényre, hogy felhasználóbarát legyen. Az említett fejezetek, működési koncepciók tematikailag vannak rendszerezve négy nagy főcsoportba: gráfelmélet, rendezés, memóriaterület és fraktálok. Ezen menüpontok könnyedén elérhetőek a főmenüből. A menüpontok rendre almenükre tagolódnak, és a tananyag által meghatározott sorrendben foglalnak helyet a munkafelületen.

A felhasználóbarát és a Pascal programozási nyelv nem igazán összeférő fogalmak, ezért kihívásnak bizonyul a kettőt egy adott programban megvalósítani. Ennek egyik alapfeltétele, hogy a billentyűzet alkalmazását kiküszöböljük, és csakis az egér nyújtotta lehetőségeket vesszük igénybe a program irányításánál. Az elvárások kielégítéséhez még az is hozzátartozik, hogy a felhasználók többsége által ismert Windows alapú munkafelületet imitálom (felül megjelenő menügomb-sorozat, domináló kék szín, megjelenő üzenetdobozok, sokak számára idegesítő konfirmálási kérés). A kezelést megkönnyebbítik a **SÚGÓ** és **ISMERTETŐ** menügombok is, amelyek az adott helyzetnek megfelelően különböző információt, útmutatót jelenítenek meg. Ezek részletes bemutatását a III. fejezetben tárgyalom.

Mivel az informatika tananyagot kívánom illusztrálni, ezért első sorban a tanárok tudják a leginkább hasznosítani a programcsaládot. Ezen felhasználási terület indokolja, igazolja a program nevét: Didaktikus pascal. Ezen kívül természetesen a diákok is hasznát vehetik tanulás közben, a különböző algoritmusok, fejezetek elsajátításánál, felelevenítésénél. Fontos kiemelni, hangsúlyozni, hogy a szimulációk lépésről lépésre futtathatóak. Ezen eljárás megkönnyíti az elsajátítás folyamatát, vagy akár a magyarázás műveletét. A továbblépést, vagy az illető alkalmazás bezárását természetesen a felhasználó dönti el, nem automatikus, és nem időfüggő. Mivel a magyarázás művelete

<sup>1</sup> **LIFO** – Last In First Out – Utolsó be, első ki

<sup>2</sup> **FIFO** – First In First Out – Első be, első ki

rendszerint video-projektor (vetítő) jelenlétét vonja maga után, ezért a továbblépés egyszerű egérgattintással valósítható meg. Ez úton nem szakad meg a gondolatmenet, nem terelődik el a figyelem, amely mindvégig az informatikai probléma megértésére összpontosul, és minden zökkenőmentesen történik.

A program révén lehetőség nyílik egyes algoritmusok (szélességi és mélységi bejárás, összefüggő komponensek, sor és verem, listák, fraktálok) saját példára (program segítségével szerkesztett) történő kipróbálására.

## **2. Program használati útmutatója – felhasználói kézikönyv**

A programcsalád összegző jellegű főmenüjét a **KOCKA.exe** kiterjesztésű fájl megnyitásával indíthatjuk el. Ezt a főmenüt akár úgy is értelmezhetjük, mint például a NERO lemezező program menüjét, ahonnan ki tudjuk választani a kívánt alkalmazást (lemezégetés, lemeztörlés, előlapkészítő, DVD film készítése).

### **2.1 Főmenü**

A főmenü vizuális megszerkesztésénél az egyszerűsége, rendezettsége törekedtem, annak érdekében, hogy a program használatát megkönnyítsem (lásd 1. ábra). A program fejléce állandó jellegű menügombokat tartalmaz, amelyek a program futása alatt mindvégig láthatók és elérhetők. Ezen úton is a Windows alapú alkalmazásokat imitálom. Minden egyes utasítás, menüpont elérése az egér segítségével történik (ráhelyezés, majd bal gomb lenyomása).

A fejléc (lásd 2. ábra) rendre a következő gombokat tartalmazza: **Főmenü, Ismertető, Súgó, Névjegy, Kilépés, Színminták**. Az aktív menügombok kék színűek, míg az inaktívak szürkék. Az aktivitási és inaktivitási tulajdonságuk helyzettől függ. Az inaktív gombokra történő kattintásnak nincs következménye.

Jobbról balra haladva az első gomb a **Színminták**. Csupán érdekességként került be a programba, és arra jó, hogy három meghatározott szín közül választhatunk, és ezt hozzárendelhetjük a háttérszínhez, vagyis meg tudjuk határozni a háttérszínt. A menügomb megnyomásával egy ablak jelenik meg (lásd 3a. ábra, 3b ábra), amelyen a három, négyzetben elhelyezett minta közül választhatunk. Ellenkező esetben, a Windowsból megszokott jobb felső sarokban elhelyezett **X** megnyomásával léphetünk vissza. A megjelenő ablak visszatérő motivuma, eleme a programnak, ugyanis az összes információ, utasítás, konfirmálási kérés ebben az ablakban jelenik meg a képernyő közepén. Rendelkezik egy kis esztétikai többlettel, hiszen szürke árnyékolással láttam el, növelve ezáltal a térhatást. A művészeti klasszicizmusra jellemző imitáció fordulatával élek, az antik remekművek helyét nálam a Windows veszi át. További érdekessége a háttérnek, hogy a szín számértéke egy kimenő, .txt kiterjesztésű szöveges állományban tárolódik, és ezáltal a program lezárása és újboli indítása után az utoljára meghatározott szín lesz a háttér színe. A módosítás pillanatában újraírodik az állomány tartalma a korszerű értékkel. Kizárólag a főmenüből érhető el.

Második menüpont a **Kilépés**. Aktiválása csak a főmenüből lehetséges, ugyanis más stádiumban inaktív állapotban van. Megnyomásával egy konfirmálási ablak jelenik meg két választási lehetőséggel: igen és nem (lásd 4. ábra). Ezek függvényében elhagyható az alkalmazás, vagy vissza lehet térni a szimulációkhoz.

Harmadik gomb a **Névjegy**. Elérése csakis a főmenüből történik, a megszokott információs ablakban megjelennek a program készítőjéről szóló adatok. Eltűnése automatikusan történik, az egyszeri elolvasáshoz elegendő időtartamot bocsájt a felhasználó rendelkezésére.

Negyedik gomb a **Súgó**. Tartalma, mint már említettem helyzettől függően változik, mindig optimális utasítást kíván közölni a felhasználóval. Az előbbiektől eltérően az almenükben is elérhető a tartalma.

Az ötödik menügomb az **Ismertető**. Arra szolgál, hogy az almenükben elhelyezett algoritmusok, fejezetek fontosabb sajátosságait fedti fel. Egyfajta elméleti betekintést biztosít a különböző részekbe. A szöveg elolvasása után Enter megnyomásával térhetünk vissza az alkalmazásba.

A hatodik, és egyben utolsó menüpont a **Főmenü**. Az elmenükből a főmenübe történő visszalépést valósítja meg.

A főmenü központjában négy egyforma négyzetet látunk. Ha az egeret ráhelyezzük valamelyikre, megjelenik a leírása (lásd 6. ábra – gráfelmélet, rendezés, memoriaterület, fraktálok). Mindaddig olvasható az információ, amíg az egér mutatója rajta található az adott négyzeten. Ujabb imitációt figyelhetünk meg. Lényege, hogy kattintás nélkül kiírja az almenü megnevezését. Természetesen a gomb megnyomásával belépünk az adott almenübe. Még a belépés pillanatát megelőzi egy kis esztétikai játék, a négyzetek átlósan elindulnak a képernyő szélei fele (lásd 7. ábra).

## 2.2 Gráfelmélet

Ezen almenü tartalmazza a legfontosabb elméletek, ciklusok, működési koncepciók vizuális megjelenítését. Hét menüpontot foglal magába (lásd 8. ábra): gráfépítő, szélességi bejárás, mélységi bejárás, összefüggő komponensek, legrövidebb utak (prím és kruskal), euler vonal, hamilton kör).

### 2.2.1 Gráfépítő

Az almenü első helyét tudhatja a magáénak, ugyanis lehetővé teszi saját tervezésű gráf megszerkesztését. Első lépés kiválasztani a csomópontok számát. Ez könnyedén elvégezhető, hiszen a megadott listából kell választani (max. 9 pont). Ezt követően a fekete munkafelületen megjelenik a kiválasztott számú csomópont egy képzeletbeli kör kerületén. A bal felső sarokban egy LCD kijelzőt imitáló barna keretben zöld vagy piros színnel különböző információs szövegek jelennek meg (a jelenlegi berajzolt élek száma vagy hibaüzenet) (lásd 9. ábra). Két csomópontot úgy kötünk össze, hogy rendre rákattintunk a két csomópontra. A bejelölést az él berajzolásának pillanatáig elszíneződés jelzi. Ha már létezik a behúzott él, akkor hibaüzenet jelenik meg, nem növeli az élek számát, és mindezt hang effektusok kísérik. Ha törölni szeretnénk, rá kell kattintani a TÖRÖL menügombra, elszíneződik, ezt követően a kívánt két pont megjelölésével meg is valósul az összekötő él eltávolítása. Minden egyes törlési művelethez működtetni kell a TÖRÖL menügombot.

Ha elkészültünk a gráffal, egyszerűen rákattintunk a KÉSZ menüpontra, átvált zöld színűre, és a felbukkanó információs ablakban megkérdi, hogy el szeretnénk-e

menteni a gráfot. Ha igennel válaszolunk, akkor egy szöveges állományba elmenti, és a következő három alkalmazást ki is lehet próbálni rajta, ha kívánjuk.

### **2.2.2 Szélességi bejárás**

A breadth first bejárás lépérről lépésre szemlélteti az algoritmus működését, melyet lefuttathatunk a beépített példára, vagy a korábban elkészített gráfra is. Kettő darab LCD-t imitáló barna keretben hasznos értékek jelennek meg, egyesek változnak ciklusról ciklusra, mások pedig csupán bővülnek. A végeredmény szemmel látható, de információ formájában az LCD-ben is megjelenik.

Az első lépésben egérrel ki kell jelölni a kezdőpontot. Egyszerűen rá kell kattintani a kívánt csomópontra.

A ciklus lépérről lépésre halad, nemcsak a végeredményt jeleníti meg, hanem az adott helyzetben a lehetséges eseteket (külön szín) és a hasonlítottat (kiemelt szín). Ezen kívül bemutatja, hogy milyen elv alapján zárja ki a lehetőségeket.

### **2.2.3 Mélységi bejárás**

A depth first bejárás az előzőhöz hasonlóan felkínálja a példák közötti választási lehetőséget, továbbá az előbbieken meghatározott eljárást (egérrel történő kezdőpontválasztás, LCD) alkalmazza az eredmény elérése érdekében.

### **2.2.4 Összefüggő komponensek**

A fekete munkafelületen szintén LCD segítségével jelenítjük meg a kívánt adatokat. Minden egyes ciklus meghatároz egy összefüggő komponenst, eltüntet az éleket, majd csak a komponensre érvényes éleket rajzolja vissza piros színnel. A továbblépéshez kattintani kell bárhova az egérrel.

### **2.2.5 Legrövidebb utak**

Az alkalmazás két eljárást mutat be (prím és kruskal), hangsúlyozván a működésbeli eltérést.

#### **2.2.5.1 Prím**

Az algoritmus kiválasztása után szöveg formájában megjelenik az eljárás rövid ismertetője, majd automatikusan végigvezet az egész műveleten. A lefutás után újból lehet nézni az alkalmazás működését (UJRA), vagy vissza lehet térni az almenübe (KILÉP). A szükséges adatok a kijelzőkön jelennek meg.

#### **2.2.5.2 Kruskal**

Az algoritmus kiválasztása után szöveg formájában megjelenik az eljárás rövid ismertetője, majd automatikusan végigvezet az egész műveleten. A lefutás után újból

lehet nézni az alkalmazás működését (UJRA), vagy vissza lehet térni az almenübe (KILÉP). A szükséges adatok a kijelzőkön jelennek meg.

### **2.2.6 Euler vonal**

Automata üzemmódban működik, menedékesen bejelöli a helyes megoldást, majd kiírja a sorozatot, menetirányt.

### **2.2.7 Hamilton kör**

Automata üzemmódban működik, menedékesen bejelöli a helyes megoldást, majd kiírja a sorozatot, menetirányt.

## **2.3 Rendezés**

Ez az almenü egy alkalmazást tartalmaz: minimumkiválasztásos rendezés. Aktiválásával megnyílik a munkafelület, amelyen egy tömb rajzolódik ki véletlesen generált számokkal feltöltve. A számok nem ismétlődnek, és minden egyes futtatásnál más-más számok jelennek meg. Visszatérő motivum az LCD, amelyen szöveges információt olvashatunk a program működéséről. A ciklusok itt is elkülönülnek, csak egy ciklus befejezése után lehet elhagyni az alkalmazást.

Megfigyelhető egy STATUS gomb is, amely három színű lehet:

- ❖ Sárga – grafikai rész működik
- ❖ Zöld – program működik
- ❖ Piros – rossz érték

A program futását különböző hang effektusok kísérik.

## **2.4 Memoriaterület**

Három elkülönülő alkalmazást tartalmaz: sor-, verem- és listaszimulációk.

### **2.4.1 Sor – LIFO**

A sor szimulálására újból tömböt használunk. A STATUS funkciója megegyezik az előző alkalmazásával. Az adatokat képviselő számok véletlesen generáltak. A PUSH gomb megnyomásával új számot helyezünk a strukturába (utolsó elem), a POP gombbal, pedig eltávolítunk egyet (mindig az elsőt). A mozgó számok csupán esztétikai szerepet töltenek be. Abban az esetben, ha a sor tele van, és be szeretnénk helyezni egy új elemet, hibaüzenet jelenik meg (túlcsordulás), ha pedig nincs elem, és szeretnénk törölni, akkor az alulcsordulás üzenet olvasható. Mindkét esetben hang effektusok növelik a hatást. A választott műveletkre érvényes információk megjelennek az LCD-n (elem értéke és pozíciója).

### **2.4.2 Verem – FIFO**

Működése megegyezik a Sor működésével.

### 2.4.3 *Lista*

Az alkalmazás kétféle létrehozást biztosít: rendezett és hagyományos. Ezek közül a rendezettet emelném ki, ugyanis a kis téglalapok a véletlenszerűen generált számokkal együtt a munkafelület bal oldalától indulnak, és megkeresik a megfelelő pozíciójukat, és elfoglalják azt. A művelet végén egy rendezett listát kapunk. Ezt követően lehet törölni a listából a TÖRÖL gomb megnyomásával és a kívánt érték kiválasztásával (egérrel). Be lehet szűrni. Ilyenkor a rákattintott elem után ékelődik be az új elem. Ha túl sok az elem, vagy nem létezik a lista, nem lehet műveletet végezni.

Minden jelentős információ megjelenik az LCD-n. A műveletek kiválasztásakor megjelenik a Pascal-ban megírt utasításrendszer, programrészlet.

## 2.5 Fraktálok

Ezen almenü négy alkalmazást foglal magába: Sárkány, Koch görbe négyszögre, Koch görbe háromszögre, Koch görbe építése.

### 2.5.1 *Sárkán, Koch görbe négyszögre és háromszögre*

Az alkalmazás elindításakor ki kell választani a rendet, majd a program automatikusan kirajzolja az alakzatot, és ugyancsak automatikusan visszalép meghatározott idő után az almenübe.

Információként megjelenik a rend értéke, és az élek száma. Az utobbi kettő esetén lehet választani a **Bemutató** menüpontot is, amely koordinátarendszer segítségével lépésről lépésre ismerteti a fraktál felépítését.

### 2.5.2 *Koch görbe szerkesztése*

Engedélyezi egy kilenc élű tetszőleges koch görbe szerkesztését. A megadott szögek segítségével lehet építeni. Miután kiválasztjuk a szöget, ki kell választani az irányát is (+ órajárással megegyező, - órajárással ellentétes). A Műveletet 8X kell megismételni. Ezt követően meghatározzuk a törésszámot, és a program automatikusan, kis időközökkel elválasztva kirajzolja az első három rendet.

## 3. Programozási háttér

Még mielőtt konkrétan rátérnék a programozási háttérre, szükséges bizonyos alapgombokat megemlíteni, megmagyarázni. Ilyen például, hogy grafikus módban a Pascal programozási nyelv 640\*480-as képernyőfelbontást engedélyez. Ez azt jelenti, hogy vízszintesen 640 pixel<sup>3</sup> helyezkedik el, függőlegesen pedig 480. Ezen kritikus információ képezi a vizuális megvalósítás alapját, ugyanis az adatok függvényében tudjuk meghatározni a különböző elemek elhelyezését (pl. Rectangle(x1,y1,x2,y2), Line(x1,y1,x2,y2), Putpixel(x,y), Circle(x,y,r) – r a sugár, szintén pixelben kifejezve),

---

<sup>3</sup> Pixel - képkocka

felíratok helyét (Outtextxy(x,y,s) – s kiírandó szöveg string formájában) a képernyőre. A grafikus mód inicializálása a következő utasítással adható meg (gd, gm: integer):

```
gd:=detect;
InitGraph(gd,gm,"");
if GraphResult<>GrOk then begin write('Grafikus hiba');
halt(1);
end;
```

Számos helyen szó esett már a program könnyű használatát megvalósító egeres irányításról. Ennek a megvalósítási háttérében egy különálló UNIT áll (mymouse.pas), amely lehetővé teszi az egér használatát a jobbra látható eljárással. Az **eger** nevet viselő eljárás a **mouseon** művelettel megjeleníti a kurzort a képernyőn, majd a **repeat** ciklus segítségével engedélyezi az egér szabad mozgását a képernyőn. A ciklus mindaddig működik, amíg le nem nyomjuk az egér bal gombját. A **getmouse** az **m** változóba eltárolja a kurzor **x** és **y** koordinátáit. A **mouseoff** természetesen eltünteti az egér fehér nyílát, és az **a** és **b** változók megkapják az **m**-ben tárolt **x** és **y** koordináták értékeit. Ezek későbbi elemzése révén különböző utasításokat tudunk működtetni. Ennek az eljárásnak köszönhetjük, hogy meg tudjuk állapítani, melyik pontban nyomtuk le az egér bal gombját. Számos alprogram hívja meg, ahol

```
uses mymouse;
...
procedure eger(var a,b:integer);
var m:mouserec;
begin
mouseon;
repeat
getmouse(m);
until m.left;
delay(250);
mouseoff;
a:=m.x;
b:=m.y;
end;
```

```
repeat
menukeszlet(8,8,1,1,1,1);
setfillstyle(1,szin);
bar(17,36,623,463);
floodfill(2,2,4);
setbkcolor(black);
negyzet;
case foeger of
1:begin mozzgat; grafelmelet; end;
2:begin mozzgat; rend; end;
3:begin mozzgat; memoria; end;
4:begin mozzgat; fraktalok; end;
5:kilepes;
6:szinmintak;
7:nejjegy;
8:sugo(0);
end;
until ki=false;
```

éppen egérrel kell meghatározni valamit, kiválasztani valamit.

A főmenü (balra)(lásd 1. kép) voltaképp a főprogramban működik, és egy **repeat** ciklussal valósul meg. Ez a ciklus mindaddig megy, amíg a **kilepes** eljárás **false**-ra nem módosítja a **ki** boolean típusú változót. Minden egyes ciklusban a **szin** változó függvényében kitölti hátteret, majd a **negyzet** eljárás meghívásával berajzolja a képernyő közepére a négy almenüt jelképező négyzetet. A program grafikai részének további elemei (keret, menükészlet) csak egyszer rajzolódnak ki, hiszen a program futása alatt nem módosulnak, és ezért még a **repeat** ciklus előtt foglalnak helyet a program strukturájában. A **foeger** egy függvény, amely visszatérít egy számot, ez a szám fogja majd meghatározni a **case** utasítással, hogy melyik almenübe lépjen be. A mozzgat

alprogram az említett vizuális effektus végrehajtására szakosodott (a négyzetek mozgatása az elfoglalt helyüktől átlósan a képernyő széle fele). A **foeger** alprogramját a következő részlet adja meg. Kiemelendő érdekessége, hogy elegendő ráhelyezni az egér mutatóját a 4 négyzet valamelyikére, és kattintás nélkül kiírja az almenü megnevezését, mindaddig amíg az egér mutatója az adott négyzeten van. A főciklusa hasonlít az előbbieken megjelenített **eger** alprogram ciklusához, egyedüli eltérés a ciklusba beékelte 4 if feltétel (piros keret). Ha a koordináták megfelelőek (*then* ág), akkor kiírja megfelelő színnel az adott megnevezést, továbbá a megfelelő színnel keretet rajzol az almenü négyzetének, tudniillik a négyzetek mind rendelkeznek egy egységes fehér kerettel. Abban az esetben, ha a koordináták nem megfelelőek (*else* ág), egyszerűen az alapszínnel ráír a felíratra és az összekötő vonalra, és végül visszarajzolja a fehér keretet.

```
function foeger:byte;
  var m:MouseRec;
      szam:byte;
  begin
    mouseon;
    repeat
      getmouse(m);
      if (m.x>285)and(m.x<305)and(m.y>205)and(m.y<225)
        then begin setcolor(4); rectangle(280,200,310,230);
          line(276,196,250,170);
          outtextxy(210,155,'Grafelmelet'); end
        else begin setcolor(15); rectangle(280,200,310,230);
          setcolor(szin); line(276,196,250,170);
          outtextxy(210,155,'Grafelmelet'); end;
      3X if (m.x>...)and(m.x<...)and(m.y>...)and(m.y<...)
    until m.left; delay(250); mouseoff;
    szam:=0;
    if (m.x>280)and(m.x<310)and(m.y>200)and(m.y<230)
      then begin szam:=1; sund; end;
    9X if (m.x>...)and(m.x<...)and(m.y>...)and(m.y<...)
    foeger:=szam;
  end;
```

Az alprogram a bal gomb lenyomását követően 10 darab if feltétel segítségével ellenőrzi a koordinátákat, majd ha a megfelelő intervallumban helyezkednek el, akkor a **szám** változó megkapja az if feltételnek megfelelő számértéket, és ezt téríti vissza. Ha rossz helyre kattintottunk, a függvény 0 térít vissza, és ebből kifolyólag nem fog semmi történni, újból lehet próbálni.

A programból való távozást a **kilep** eljárás valósítja meg. Egy egyszerű kilépés helyett egy Windows-os kilépésre emlékeztető konfirmációs ablakocska jelenik meg, ha netán a felhasználó időközben meggondolta volna magát. Ha itt az IGEN-re kattint, akkor valóban kilép a programból, hiszen a globális **ki** változó értéke *false*-ra vált, ellenkező esetben az ablakocska eltűnik, és tovább lehet folytatni a böngészést.

```

procedure menukeszlet(a,b,c,d,e,f:byte);
begin
  setcolor(a);
  outtextxy(32,22,'Fomenu');
  setcolor(b);
  outtextxy(102,22,'Ismerteto');
  setcolor(c);
  outtextxy(202,22,'Sugo');
  setcolor(d);
  outtextxy(270,22,'Newjegy');
  setcolor(e);
  outtextxy(350,22,'Kilepes');
  setcolor(f);
  outtextxy(425,22,'Szinmintak');
end;

```

A 2.1 fejezet második paragrafusa szerint a menübára aktív gombjai kék színűek, míg az inaktívak szürkék. Mivel helyzettől függenek, ezért egy eljárás (menukeszlet) segítségével tudjuk meghatározni a gombok színét. Számos más alprogram hasznosítja, elegendő meghíváskor 8-kal (szürke) vagy 1-el (kék) feltölteni. A következő két eljárásnak információközvetítő szerepük van: sűgó és ismertető. Mindkettő egy *byte* típusú bemeneti paraméterrel rendelkezik, amelyek **case** típusú eljárással kerülnek mérlegelésre. Az **ismerteto** szöveges állományokból olvassa ki a megjelenítésre szánt hosszabb szöve-

get, míg a **sugo**, igénybe veszi a **mellekmenu** alprogram által biztosított ablakocskát, és ezen jelenít meg egy párszavas útbaigazítót.

A **mellekmenu** a képernyő közepére egy halványkék alapú téglalapot rajzol ki, szürke árnyékolással. A program strukturájában számos helyen meghívjuk, a szöveg csupán utólag kerül fel az információs ablakocskára.

A program használatakor feltűnik, hogy mikor megjelenik egy már említett információs ablakocská, és eltűnik, az alatta lévő grafikai felület nem sérül, pontosabban visszáll az ablak megjelenése előtti helyzet. Normál esetben ez nem lehetséges, mivel tudjuk, hogy a képernyőre felrajzolt grafikus ábra statikus, nem módosítható csupán ráfestéssel, és ezáltal a régi felület megsemmisül. Ezen problémát sikerült megoldanom dinamikus változók igénybevételével. A logikus gondolkodás első sorban azt sugallja, hogy képszerűen el kellene menteni az adott grafikus felületet, kiírni az információt, majd a képet visszahelyezni az eredeti helyére. Ezt a megoldási lehetőséget valósíthatjuk meg

```

var meret:word;
    p:pointer;
...
meret:=imagesize(x1,y1,x2,y2);
getmem(p,meret);
getimage(x1,y1,x2,y2,p^);
putimage(x,y,p^,normalput);

```

dinamikus változókkal a következő programrészlet segítségével (balra). az **imagesize**-al kijelöljük a képrészlet méretét, majd a **getmem** és **getimage** paranccsal elmentjük a **p** dinamikus változóban. Végül a **putimage**-el visszahelyezzük az **x,y** meghatározott helyre. Ezen eljárás segítségével azt a látszatot kelti a program, hogy valójában csak eltűnik az információs ablakocská (2. kép). A térhatást még az is növeli, hogy a megjelenő ablak árnyékkal

is rendelkezik, és végső fokon, olyan mintha a grafikus felület fölött megjelenne, majd eltűnne. Valójában az történik, hogy az ablak megjelenésével megsemmisíti az adott képrészletet, de viszont ez el van mentve, s ezáltal könnyedén visszahelyezhető. Kiváló példája a felhasználásának a sor és verem szimulációknál mutatkozik. Bármelyik pillanatban igénybevehető a sűgó nyújtotta leírás, és az automatikusan eltűnő ablak alatt a grafikai felület sértetlen marad, és tovább folytatható a szimuláció. Ezen kívül a program majdnem minden pontján mutatkozik az említett képmentési művelet.

Minden egérekattintást egy hang kísér, amelyet a **sund** alprogram valósít meg. Nem rendelkezik paraméterlistával. Kivételt képeznek a rossz kattintások, illetve a kiemelt helyzetek.

Számos olyan ciklus, megoldás került a dolgozatomba, amelyek csupán az esztétikai, grafikai részhez járulnak hozzá, de ugyanakkor növelik az alkalmazás kezelhetőségét. Ilyen például a már említett háttérszín-váltás, megjelenő információs ablakok, de ugyanakkor ezek programozási kihívások is. A **torol** alprogram arra szolgál, hogy ne egyből törölje a képernyő tartalmát, hanem menedékesen. Működési hátterében egy **ciklus** áll, amely **delay(...)** késleltetéssel a paraméterként kapott színnel vízszintes vonalakat húz, 1 pixel eltéréssel, és ezáltal teljesen lefedi a meghatározott területet. Itt akár megemlíthetjük a **sorkeret** (sor és rendezés) és **veremkeret** (verem) eljárásokat is, amelyek pixelenként rajzolják meg a kereteket.

A program elindításának pillanatában a program megadott helyre kirajzol egy négyzetet, halványkék alappal és fehér szegővel. Ezt elmenti egy dinamikus változóba, majd törli a képernyő tartalmát, és elindul a program főmenüje. Ez a lépés a felhasználó számára nem észlelhető. Az almenük négyzeteiről van szó, amelyeket a putimage paranccsal és megfelelő koordinátákkal bárhova, akárhányszor felrajzolhatjuk. Ezen eljárás megkönnyíti a programozó feladatát.

A dinamikus változókat hasznosítani tudjuk a listás feladatoknál. Itt több vonatkozásban jelentkeznek (mint képtároló változók és mint információs változók). A képszintézis bonyolultabb folyamat. Első lépésben rendelkezünk egy eljárással, amely meghatározott színben megrajzolja egy adat keretét, adott helyre. Ezt követően elmentjük egy dinamikus változóba és a kiíratási koordinátákat ciklus segítségével módosítjuk. Ennek köszönhetően egy mozgókép jön létre. Mindig egy pixellel nagyobb képet mentünk el, hiszen így kapunk egy pixelnyi vastagságú keretet, és nem kell törödnünk a visszamaradó foltokról, elmosódásokról.

```
procedure koordinata(a,b,c:integer);
var i,j,m:integer;
begin
i:=a-80; j:=b-80;
while i<=a+80 do begin
    putpixel(i,b,c);
    putpixel(a,j,c);
    i:=i+2; j:=j+2;
    delay(2);
end;
if c<>0 then c:=14;
i:=a-50; j:=b-50; m:=a+50;
while i<=a+50 do begin
    putpixel(i,j,c);
    putpixel(m,j,c);
    i:=i+2; j:=j+2; m:=m-2;
    delay(2);
end;
end;
```

Az almenük rendre jelképesen főprogramokká emelkednek, hiszen terjedelem szempontjából a legnagyobb eljárások. Hosszú felépítésük miatt igencsak nehéz dolgozatban megjeleníteni őket, viszont kisebb jelképes ábrázolással kiválthatjuk őket. A megszokott **repeat** ciklus segítségével az almenük is különálló programként futnak, csupán az almenü megnyomásával lehet kilépni belőlük, és visszatérni a főmenübe.

A gráfelmélettel foglalkozó almenü az iskolában tanított ciklusok, programok segítségével van megvalósítva. Ezekbe a ciklusokba ágyazódnak be a különféle grafikai megjelenítésre szakosodott elemek, eljárások, függvények. Ennek eredményeképpen mutatható be a gráfelmélettel foglalkozó tananyagrészt. A gráfok kirajzolására két eljárást alkalmazhatunk. Az egyik a csomópontok számának függvényében egy képzeletbeli kör sugarára

felrajzolja a csomópontokat, a másik pedig az őket összekötő vonalakat. Ezen alprogramok módosításával érhetjük el, hogy csak adott pontok színeződjenek el, vagy akár az összegüggő komponensek esetében csak az egy komponenshez tartozó vonalakat rajzolba be.

```

procedure vonalak(n,a:byte);
var i,j:byte;
    x1,y1,x,y:integer;
begin
for i:=1 to n do
  for j:=1 to n do
    if pp[i,j]<>0 then begin
      x:=x0+round(r*cos(2*pi*i/n));
      y:=y0+round(r*sin(2*pi*i/n));
      x1:=x0+round(r*cos(2*pi*j/n));
      y1:=y0+round(r*sin(2*pi*j/n));
      setcolor(a);
      line(x,y,x1,y1);
    end;
  end;
end;

procedure korok(n:byte);
var i:byte;
    x,y:integer;
    s:string;
begin
for i:=1 to n do begin
  alfa:=2*pi*i/n;
  x:=x0+round(r*cos(alfa));
  y:=y0+round(r*sin(alfa));
  setbkcolor(black);
  setcolor(red);
  circle(x,y,10);
  setfillstyle(4,black);
  floodfill(x,y,red);
  setcolor(white);
  str(i,s);
  outtextxy(x-3,y-2,s);
end;
end;

```

A rendezés almenü a minimum-kiválasztásos rendezést tartalmazza. A programnak a különlegességét a grafikai elemek adják meg. Az almenühöz kapcsolódó eljárások a számok különböző irányba történő mozgatását valósítják meg.

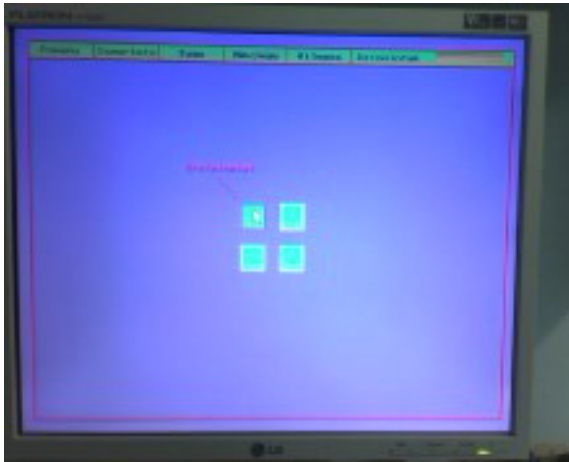
A fraktálok almenü esetében két érdekességet emelnék ki: koordináta rendszer segítségével történő bemutató és fraktal-építő. Az első valójában egyetlen alprogramnak köszönhető (**koordinata**), x,y koordináták függvényében megrajzolja a koordináta rendszert, ahol x,y az origó helyét adják meg.

A fraktal-építő esetében egy megszokott koch-görbe kirajzolására szakosodott alprogramot láthatunk, amelyben egyedüli módosítás, hogy a 8 szög értéke nem előre, hanem felhasználó által meghatározott. Az **ir1..8** változó tartalmazza az iránytényezőt, az **alfa1..8** pedig a szögek értékét. Ezek meghatározására menürendszer áll a felhasználó rendelkezésére.

#### 4. Következtetések

A program lehetőséget nyújt kiegészítésre is. A tananyag többi ciklusa, programja is beleintegrálható. Kiegészíthető még több személyes szimulációval. Rendszerigénye minimális: CPU 600Mhz, Memoria 128Mb, Videokártya: 8Mb, Tárhely: min. 500Kb.

## 5. Függelékek



1. ábra



2. ábra



3a. ábra



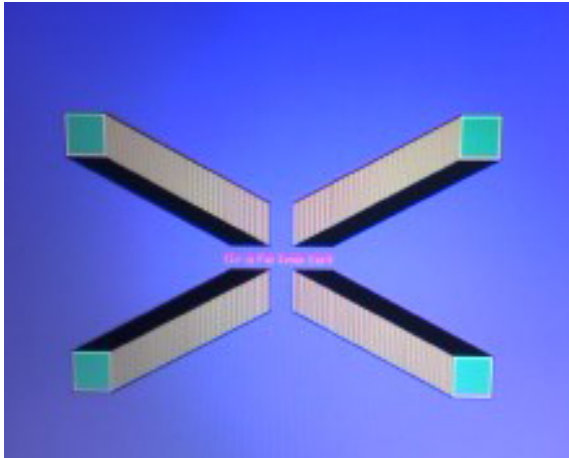
3b. ábra



4. ábra



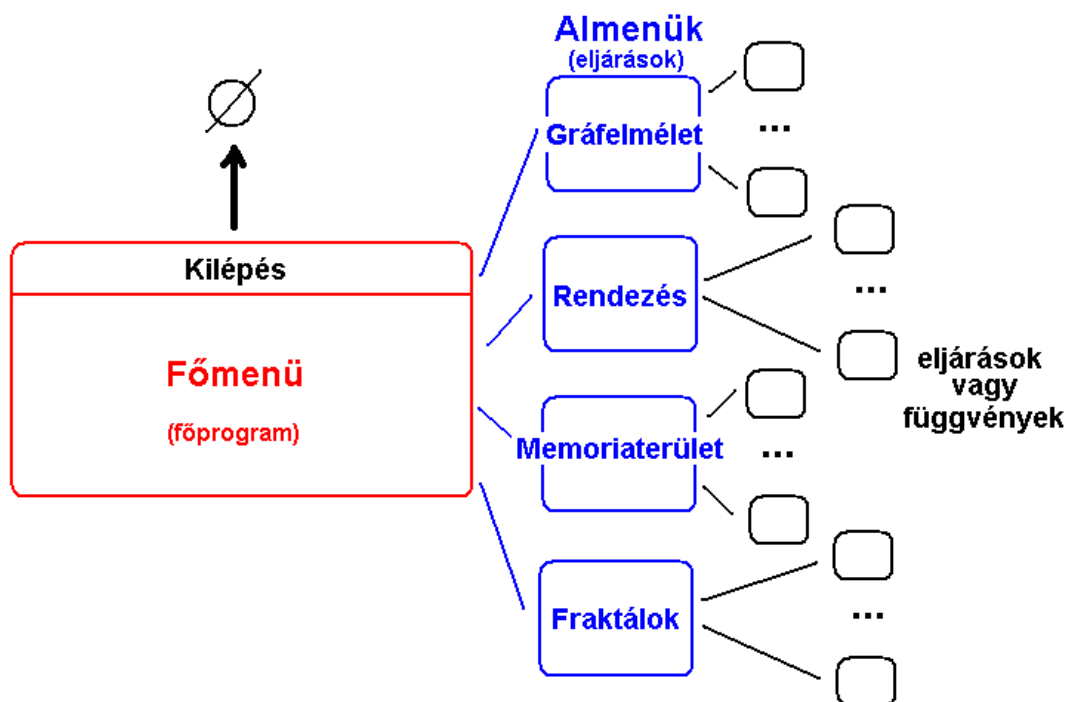
6. ábra



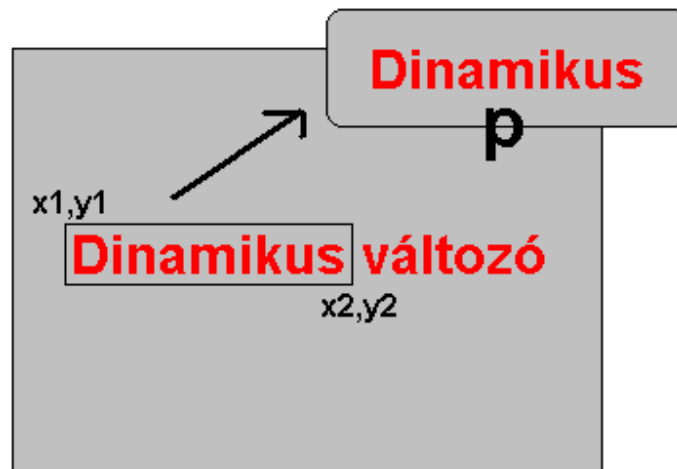
7. ábra



8. ábra



1. kép



2. kép

### ***Forrásanyag:***

1. Ignát Judit Anna – Incze Katalin – Jakab Irma Tünde: Informatika tankönyv a IX. osztály számára – Intenzív
2. Ignát Judit Anna – Incze Katalin – Jakab Irma Tünde: Informatika tankönyv a XI. osztály számára – Intenzív