

GESTIUNEA ECRANULUI ÎN MOD GRAFIC

INTRODUCERE

Un *program grafic* generează o *ieșire grafică* mai curând decât o ieșire text. Conceptual, *ieșirea text* poate fi considerată drept o ieșire către un dispozitiv ce constă într-o matrice dreptunghiulară de caractere folosite pentru a afișa linii de text. Spre deosebire de aceasta, *ieșirea grafică* poate fi gândită ca ieșire la un dispozitiv (fereastra grafică) constând dintr-o suprafață dreptunghiulară de **pixeli**¹; în plus, fiecărui pixel i se poate seta o anumită **culoare, specifică**.

Ieșirea grafică poate consta în desene și/sau imagini. În plus, ieșirea grafică poate include și text; în acest din urmă caz, ca și particularitate a modului grafic, avem faptul că fiecare literă reprezintă, în sine, tot o imagine (mai mică sau mai mare, funcție de dimensiunea aleasă a fontului).

În modul grafic, numărul de pixeli depinde de **adaptorul grafic** și se numește **rezoluție**. O rezoluție (*nr. rânduri x nr. coloane*) este cu atât mai bună cu cât este mai mare (a se vedea *Anexa 2 (1)*).

Pentru a afla informații privind adaptorul grafic al stației dvs. de lucru (în particular, **tip adaptor, rezoluție, memorie RAM, etc.**) selectați: **Start/All Programs/Accessories/System Tools/System Information/Components/Display**.

Fereastra grafică, creată și afișată de aplicația grafică, constă într-un tablou bi-dimensional, cu $n*m$ pixeli. Aceasta înseamnă că, pe ecran, se pot afișa m linii a câte n pixeli fiecare. Lățimea ferestrei (**W**) este dată de numărul de pixeli/linie iar înălțimea sa (**H**) este dată de numărul de pixeli/coloană. Fiecare pixel este specificat prin poziția coloanei și, respectiv, liniei sale, iar acestea mai poartă numele și de *coordonatele pixelului*. Coloanele se numeroatează de la stânga la dreapta iar liniile de sus în jos; astfel, coordonatele **(0,0)** corespund pixelului din colțul stânga-sus al ferestrei iar coordonatele **(W-1, L-1)** corespund pixelului din dreapta-jos al ferestrei (vezi *Fig. 1*). Ca și în cazul sistemului de coordonate cartezian, x reprezintă distanța orizontală de la punctul de origine **O** iar y reprezintă distanța pe verticală; facem mențiunea că, spre deosebire de sistemul cartezian, y crește pe măsură ce coborâm (nu pe măsură ce urcăm!!).

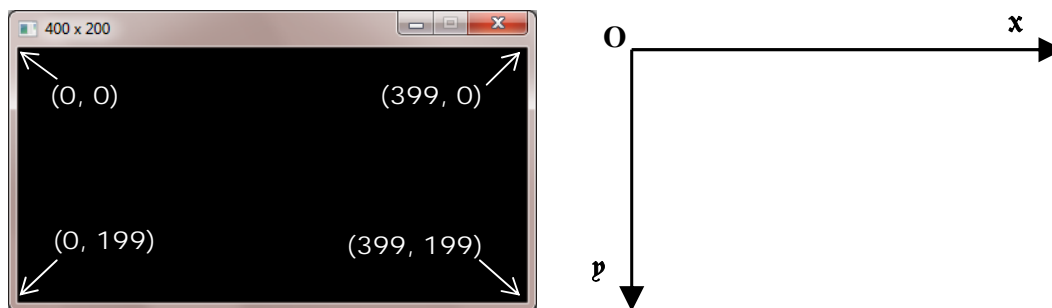


Figura 1: Exemplu de fereastră grafică

Pentru gestiunea ecranului în mod grafic se pot utiliza peste 60 de funcții standard aflate în biblioteca grafică **Borland Graphics Interface** (BGI) pentru programe DOS. Acestea au prototipurile în fișierul **<graphics.h>**. Programele grafice vor utiliza, din acest motiv, directiva preprocesor **#include<graphics.h>** ce dispune includerea în programul nostru sursă a fișierului header "graphics.h". În acest capitol indicăm funcțiile mai importante ce permit gestiunea ecranului în mod grafic, funcții ce folosesc pointeri de tip *far* (32 de biți).

Biblioteca grafică **WinBGI** (*Windows BGI – cu mouse*) reprezintă un port al librăriei BGI pentru Microsoft Windows. Aceasta aduce în plus un suport pentru lucrul cu mouse-ul în cazul aplicațiilor Windows, **pentru controlul ferestrelor grafice, precum și suport pentru utilizarea ferestrei grafice ca un C++ ostream**. De asemenea, oricare dintre funcțiile *winbgi* ce utilizează culori poate utiliza și **culorile RGB** adițional la paleta

¹ Modul grafic presupune că ecranul este format din "puncte luminoase" (elemente de imagine/pixeli).

BGI de 16-culori. Funcțiile grafice Windows disponibile vor fi indicate în continuare prin indicativul **win** iar utilizarea lor implică includerea aceleiași biblioteci *graphics.h*.

Instrucțiuni de instalare & configurare a librăriei WinBGI cu Dev-C++

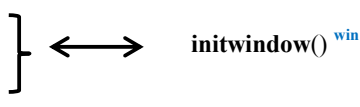
1. După instalarea mediului Dev-C++ descărcați fișierul header [graphics.h](#) pe care îl veți include în subdirectorul Dev-Cpp/MinGW32/include.
2. Descărcați [codul sursă](#) pentru librăria **WinBGI** și compilați-l pentru a construi fișierul **libbgi.a**. Pentru aceasta, în mediul Dev-C++ veți crea un proiect cu numele **libbgi**; dintre templat-urile puse la dispoziție de mediu veți selecta opțiunea **Basic**, iar în cadrul acesteia tipul **Static Library**; acesta va genera, la final, o *bibliotecă statică* (.a) având același nume cu cel al proiectului (în cazul nostru, **libbgi.a**).
3. Includeți biblioteca **libbgi.a** astfel creată în subdirectorul Dev-Cpp/MinGW32/lib.
4. Pentru a putea utiliza funcțiile grafice, un ultim pas se impune a fi făcut atunci când veți crea un proiect grafic. Editorul de legături va trebui instruit să realizeze legături într-o serie de librării. Pentru aceasta, din meniul **Project** se va selecta **Project Options** (alternativ, se poate utiliza **Alt+P**). Din fereastra asociată se alege tab-ul **Parameters**, iar în aria dedicată editorului de legături se vor introduce următoarele informații:

-lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32

FUNCȚII GRAFICE

1. Setarea modului grafic

Funcțiile discutate:

- **detectgraph(...)**
 - **initgraph (...)**
 - **graphresult(...)**
 - **grapherrormsg(...)**
 - **getcurrentwindow()** **win**
 - **setcurrentwindow()** **win**
 - **closegraph(...)**
- 

Prototipurile funcțiilor:

```
void detectgraph(int *graphdriver, int *graphmode);  
void initgraph(int *graphdriver, int *graphmode, char *pathtodriver);  
int initwindow( int width, int height, const char* title="Windows BGI", int left=0, int top=0, bool  
dbflag=false, bool closeflag=true);  
  
int graphresult(void);  
char *grapherrormsg(int errorcode);  
int getcurrentwindow(void);  
void setcurrentwindow( int window);  
void closegraph(void);
```

Modul grafic se inițializează cu ajutorul funcției "**initgraph**". Această funcție poate fi folosită singură sau împreună cu o altă funcție, numită "**detectgraph**" – o funcție ce detectează adaptorul grafic al sistemului și determină parametrii acestuia:

- ✓ *graphdriver*, un parametru ce pointează spre zona în care se păstrează valoarea (a se vedea *Anexa 2 (1)*) ce definește driver-ul² corespunzător adaptorului grafic detectat, și

² În sistemele de operare, **driver-ul** este un program ce acționează ca o interfață între o anumită componentă hardware și software; el transmite către componenta hard comenzile ce vin, prin intermediul S.O., de la diferite programe. În cazul nostru driver-ele grafice (fișierele **.BGI**) se regăsesc în subdirectorul BGI al directorului BORLANDC.

- ✓ *graphmode*, ce definește modul grafic cu cea mai mare rezoluție pentru adaptorul detectat.

Versiunea winbgim a funcției **detectgraph** returnează **VGA** pentru *graphdriver* și **VGAHI** pentru *graphmode*, indiferent de hardware-ul mașinii. Cu toate acestea, ecranul nu este în mod necesar 640 x 480.

(1) Apelul funcției "detectgraph" este urmat de apelul funcției "initgraph". Aceasta din urmă setează modul grafic în conformitate cu parametrii stabiliți de apelul prealabil al funcției "detectgraph" (vezi **Ex.1**). (2) În lipsa apelului acestei funcții, în lista parametrilor funcției **initgraph** se pot indica, direct de către utilizator, un driver grafic particular și un mod grafic particular (vezi **Ex.2**). Din această perspectivă:

- ✓ **graphdriver* reprezintă un număr întreg ce indică driver-ul grafic ce se va utiliza. Acestui parametru i se poate da o valoare reprezentând o constantă de tipul enumerării drivere-lor grafice așa cum apare definită în `graphics.h` (a se vedea **Anexa 2(1)**);
- ✓ **graphmode* este un număr întreg ce indică *modul grafic inițial* (rezoluția) – excepție face cazul când **graphdriver* = DETECT; în acest ultim caz, **graphmode* este setat de **initgraph** la cea mai mare rezoluție disponibilă pentru driver-ul detectat. Valorile pe care le poate lua **graphmode* sunt, de asemenea, indicate în **Anexa 2(1)**. (**Atenție:** *graphdriver* și *graphmode* trebuie să ia valori dintre cele valide, specificate în tabelele din anexa; în caz contrar se vor obține rezultate imprevizibile).

(3) Alternativ, se poate cere funcției **initgraph()** să determine, prin autodetecție, adaptorul video atașat la momentul execuției și să aleagă/seleceze driver-ul corespunzător acestuia (vezi **Ex.3**). În cazul autodetecției funcția **initgraph**:

- apelează **detectgraph()** pentru a defini parametrii implicați ai adaptorului grafic al sistemului;
- resetează toate setările grafice la valorile lor implicite (poziție curentă, paletă, culoare, viewport³ etc.) și
- resetează **graphresult** la 0 (*graphresult* returnează codul de eroare pentru ultima operație grafică care a raportat o eroare și resetează nivelul de eroare la **grOk**).

În toate cele 3 situații, parametrul *pathdriver* reprezintă un pointer spre șirul de caractere ce definește calea către subdirectorul BGI unde se vor căuta driverele grafice (de ex., "C:\BORLANDC\BGI"). Pentru început, **initgraph** se uită în calea specificată în *pathdriver*; ulterior (dacă această cale nu este specificată - cu alte cuvinte, *pathdriver* este null) **initgraph** va căuta **fișierele driver** (.BGI) în directorul curent. Tot aici **settextstyle()** va căuta fișierele cu font-urile pentru caractere (*CHR).

Funcția **initgraph** setează întotdeauna codul de eroare internă: 0 – în caz de succes, iar în caz de eroare, **graphdriver* ia valoarea -2, -3, -4 sau -5 având următoarele semnificații:

Constant Name	Number	Meaning
grNotDetected	-2	Nu se poate detecta un card grafic
grFileNotFound	-3	Nu poate fi găsit driver-ul
grInvalidDriver	-4	Driver invalid
grNoLoadMem	-5	Insuficientă memorie pentru a încărca driver-ul

Versiunea *winbgim* a funcției **initgraph** utilizează parametrii săi doar pentru a determina mărimea ferestrei ce urmează a fi create. Astfel, în **Ex. 3**, indicarea de către utilizator a parametrilor CGA pentru driver și CGACO pentru modul grafic va crea o fereastră de 320 x 200 pixeli. Ca o soluție alternativă, utilizatorul poate apela **initwindow(width, height)** în locul funcțiilor *initgraph()* și, respectiv, *detectgraph()*. Funcția inițializează sistemul grafic prin deschiderea unei ferestre grafice de dimensiunile specificate (vezi **Ex.4**). Primii doi parametri (*lățimea* și *înălțimea*) trebuie specificati în mod obligatoriu, restul parametrilor având valori implicite:

- ✓ *title* reprezintă titlul afișat pe bara de titlu a ferestrei (valoarea implicită fiind "Windows BGI");
- ✓ *left* și *top* sunt parametrii ce determină coordonatele ecranului aferente colțului stânga-sus al ferestrei grafice;

³ Un viewport (o fereastră) reprezintă o regiune poligonală a ecranului utilizată pentru a afișa o porțiune din imaginea totală ce se dorește a fi afișată.

- ✓ *dbflag* este parametrul ce determină dacă dubla-buffer-are pentru fereastră este setată automat;
- ✓ *closeflag*, atunci când are valoarea TRUE, asigură posibilitatea de a închide întreg programul printr-un click pe butonul de închidere al ferestrei.

Variante de cod pentru inițializarea sistemului grafic:		
Ex.1:	<pre>int gd, gm; detectgraph(&gd, &gm); initgraph(&gd, &gm, "c:\\borlandc\\BGI");</pre>	<pre>// detectarea adaptorului grafic al // sistemului si determinarea para- // metrilor acestuia (driver-ul, res- // pectiv, modul grafic), urmată de // inițierea sistemului grafic cu pa- // rametrii astfel găsiți</pre>
Ex.2:	<pre>int gd, gm; gd = 1; gm = 0; initgraph(&gd, &gm, "c:\\borlandc\\BGI");</pre>	<pre>// constanta ce def. driver-ul CGA // constanta ce def. modul CGACO // setarea modului grafic corespun- // zător unui adaptor grafic CGA, // cu rezoluția 320 x 200 puncte – // parametri indicați de utilizator.</pre>
Ex.3:	<pre>int gd = DETECT, gm; initgraph(&gd, &gm, "c:\\borlandc\\BGI");</pre>	<pre>// cerere pentru autodetecție // setarea modului grafic cu para- // metrii implicați ai adaptorului // grafic al sistemului.</pre>
Ex.4:	<pre>initwindow(1000, 400, "Primul meu exemplu!!!");</pre>	<pre>// deschide o fereastră grafică de // 1000 x 400 pixeli</pre>

Spre deosebire de versiunea originală a funcției *initgraph()* ce permitea crearea într-un program doar a unei singure ferestre grafice, funcția *initwindow()* permite crearea de ferestre grafice multiple. Valoarea returnată în acest ultim caz reprezintă ID-ul (un număr întreg, unic) asociat fiecărei ferestre create. Întotdeauna, ultima fereastră grafică creată reprezintă fereastra curentă.

Erorile survenite în gestiunea în mod grafic a ecranului pot fi puse în evidență cu ajutorul funcției "**graphresult**". Funcția returnează codul ultimei erori care a apărut înaintea apelului funcției *graphresult()* sau valoarea constantei simbolice *grOk*, dacă nu au fost erori. Constanta *grOk* este definită în fișierul *graphics.h* și ea are valoarea zero. Codurile de eroare au valori întregi negative, în domeniul [-15,0] (a se vedea *Anexa 2(2)*). De reținut aici este faptul că variabila menținută de *graphresult* este resetată la 0 după ce s-a realizat apelul funcției *graphresult*. Din acest motiv, se indică stocarea valorii returnate de către funcție într-o variabilă temporară și abia apoi testarea ei.

Mesajul de eroare poate fi decodificat cu ajutorul funcției "**grapherrormsg**" al cărei parametru, *codeerror*, reprezintă valoarea returnată de funcția *graphresult*. Funcția *grapherrormsg* returnează adresa zonei de memorie în care este memorat textul de eroare.

După apelul cu succes al funcției *initgraph()* se pot utiliza celelalte funcții standard de gestiune grafică a ecranului. Din modul grafic se poate ieși apelând funcția **closegraph()**.

Apelul funcției **closegraph()** închide sesiunea grafică, asigură dezalocarea memoriei alocate de către sistemul grafic (memorie alocată pentru drivere, font-uri, buffer-ul intern) și determină, totodată, revenirea ecranului la modul în care acesta s-a aflat înaintea apelului funcției *initgraph* (a se vedea *Ex.5* de mai jos).

Ex.5:	<pre>#include<stdio.h> #include<conio.h> #include<graphics.h> int main(void) { int gd = DETECT, gm, errorcode; initgraph(&gd, &gm, " "); errorcode = graphresult(); if(errorcode!=0)</pre>	<pre>// cerere pentru autodetecție // inițializarea modului grafic // citirea rezultatului inițializării // testarea apariției unei erori și, în</pre>
-------	---	--

<pre> { printf("\nEroare grafică: %s\n", grapherrormsg(errorcode)); printf("\nApăsați o tastă ..."); getch(); exit(1); } // aici se insereaza codul programului grafic getch(); closegraph(); printf("\nAm revenit la modul text!\n"); printf("\nApasati o tasta..."); getche(); return 0; } </pre>	<pre> // caz afirmativ, tratarea ei. // funcție definită în graphics.h⁴ // închiderea sistemului grafic // afișare mesaj în fereastra text </pre>
--	--

Versiunea *winbgim* a funcției **closegraph**:

```
void closegraph( int wid=ALL_WINDOWS); win
```

are un parametru opțional numit *wid* ce reprezintă ID-ul ferestrei ce se dorește a fi închisă. ID-ul este o valoare returnată de *initwindow()*. Parametrul *wid* poate avea una din cele două valori constante:

- **CURRENT_WINDOW** sau **-1**, ce determină închiderea exclusivă a ferestrei grafice curente;
- **ALL_WINDOWS** sau **-2**, valoare implicită ce determină închiderea tuturor ferestrelor grafice deschise;

Dacă fereastra curentă este închisă atunci nu mai există o fereastră curentă și, deci, nu mai pot fi realizate alte operații de desenare până când, fie se creează o nouă fereastră grafică, fie se setează o fereastră grafică curentă prin apelul funcției **setcurrentwindow**.

Apelul funcției **getcurrentwindow()** returnează ID-ul ferestrei curente (în care se realizează toate operațiile grafice) iar **setcurrentwindow()** este funcția ce schimbă fereastra curentă cu fereastra specificată drept parametru.

Alte funcții ce pot fi utilizate aici sunt și:

- | | | |
|-------------------------|------------------------|------------------------|
| - setgraphmode | - graphdefaults | - getmodename |
| - restorecrtmode | - getgraphmode | - getdrivername |
| | | - getmoderange |

O funcție dedicată interacțiunii cu utilizatorul este funcția cu prototipul:

```
int kbhit(void);
```

Această funcție returnează valoarea 0 dacă nu există caractere ce așteaptă să fie citite și o valoare nenulă în cazul în care există astfel de caractere (s-a apăsă o tastă).

2. Gestiunea culorilor și a pattern-urilor

Adaptoarele grafice sunt prevăzute cu o zonă de memorie în care se păstrează date specifice gestiunii ecranului. Această zonă poartă denumirea de **memorie video**. În cazul adaptoarelor color, unui pixel îi corespunde o culoare, informație ce se păstrează pe biți în memoria video. În cazul adaptoarelor monocrom pixelul are o nuanță de gri.

⁴ Funcția *getch()* este disponibilă și în *implementarea winbgim* (în acest caz includerea header-ului *graphics.h* nu mai face necesară și includerea fișierului *conio.h*). Apelul funcției determină funcționalitatea cunoscută (citirea unui caracter de la tastatură și returnarea codului său ASCII), însă, cu diferența majoră că această implementare a funcției lucrează strict cu fereastra grafică!!! Astfel, utilizarea acestei funcții după apelul **closegraph()** generează următorul mesaj de eroare: **"Drawing operation was attempted when there was no current window"**. Pentru tastele speciale funcția *getch()* returnează, într-o primă etapă valoarea ASCII 0, pentru ca la un al doilea apel funcția să returneze valorile (sau constantele simbolice asociate): **KEY_UP** (72), **KEY_PGUP** (73), **KEY_LEFT** (75), **KEY_RIGHT** (77), **KEY_DOWN** (80), **KEY_PGDN** (81).

Gestiunea culorilor este dependentă de tipul de adaptor grafic (a se vedea *Anexa 2(2)*). Culorile se codifică prin numere întregi iar mulțimea culorilor ce pot fi afișate simultan pe ecran se numește *paletă*. Culorile din componența unei palete pot fi modificate de utilizator prin intermediul funcțiilor standard. În acest context se pot utiliza:

- | | | |
|------------------------|----------------------------|-------------------------|
| - setpalette | - getpalette | - getmaxcolor |
| - setallpalette | - getdefaultpalette | - getpalettesize |

Câteva din funcțiile de gestiune a culorilor pentru desenare sunt prezentate mai jos:

- | | |
|---|--|
| <code>void setbkcolor(int <i>culoare</i>);</code> | - schimbă culoarea de fond |
| <code>int getbkcolor(void);</code> | - obține culoarea de fond |
| <code>int getdisplaycolor(int <i>culoare</i>);</code> | - obține culoarea actuală afișată pe monitor |
| <code>void setcolor(int <i>culoare</i>);</code> | - modifică culoarea pentru desenare |
| <code>int getcolor(void);</code> | - obține culoarea pentru desenare. |
- 1) În prototipurile de mai sus "culoare" reprezintă un index în tabloul care definește paleta (și el ia valori cuprinse între 0-15 în cazul aplicațiilor care vor urma). De asemenea, cele 16 culori BGI pot fi indicate și prin nume simbolice (BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY, LIGHTBLUE, LIGHTGREEN, LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW, WHITE).
 - 2) Culorile pentru Windows BGI includ, pe lângă culorile BGI de mai sus, culorile indicate printr-o combinație a componentelor **red**, **green** și **blue** (roșu, verde și albastru); în acest sens se apelează funcția **COLOR(r,g,b)**. Fiecare dintre argumentele r,g,b trebuie să fie un număr în domeniul 0 ÷ 255. Dacă se creează o astfel de culoare, aceasta poate fi folosită ca argument în orice funcție BGI ce așteaptă informația de culoare. Alte funcții (macro-uri) RGB sunt: RED_VALUE(v), GREEN_VALUE(v), BLUE_VALUE(v), IS_BGI_COLOR(v), IS_RGB_COLOR(v).

Pe lângă setarea culorii pentru desenare alte setări pot include culoarea și pattern-ul de umplere a suprafețelor, precum și setarea caracteristicilor liniei de desenare. Funcțiile dedicate pentru acestea din urmă sunt:

- | | |
|--|---|
| <code>void setfillstyle(int <i>pattern</i>, int <i>culoare</i>);</code> | - setează pattern-ul (vezi <i>Anexa 2(4)</i>) și culoarea de umplere curente ⁵ . |
| <code>void setfillpattern(char *<i>upattern</i>,
int <i>culoare</i>);</code> | - este similară funcției setfillstyle cu deosebirea că ea este utilă în setarea unui pattern 8x8 definit de utilizator prin intermediul pointer-ului <i>upattern</i> ; acesta pointează către o secvență de 8 bytes, fiecare byte corespunzând la 8 pixeli din pattern (fiecare bit cu valoare 1 corespunde unui pixel afișat pe ecran). |
| <code>void getfillsettings(struct fillsettingstype
*<i>fillinfo</i>);</code> | - umple în structura fillsettingstype indicată prin pointerul fillinfo (a se vedea <i>Anexa 2(3)</i>) informația despre culoarea și pattern-ul de umplere curente. |
| <code>void setlinestyle(int <i>stilLinie</i>, unsigned
<i>upattern</i>, int <i>grosime</i>);</code> | - setează stilul pentru toate liniile desenate cu funcțiile de desenare (a se vedea <i>Anexa 2, (5)</i>). Atunci când <i>stilLinie</i> are valoarea USERBIT_LINE (4) prin valoarea <i>upattern</i> se indică șablonul pe 16 biți, definit de utilizator și folosit în desenearea liniei (de ex., 0xFFFF, 0x3333). |

3. Starea ecranului

- | | |
|---------------------------------------|--|
| <code>int getmaxx(void)</code> | - returnează coordonata maximă pe orizontală; |
| <code>int getmaxy(void)</code> | - returnează coordonata maximă pe verticală; |
| <code>int getx(void)</code> | - returnează poziția pe orizontală a pixelului curent; |
| <code>int gety(void)</code> | - returnează poziția pe verticală a pixelului curent. |

⁵ Stilul EMPTY_FILL indică o umplere ce folosește culoarea de background.

4. Gestiunea textelor

Funcțiile dedicate:

- **settextstyle (...)**
- **settextjustify (...)**
- **gettextsettings (...)**
- **setusercharsize (...)**
- **outtext (...)**
- **outtextxy (...)**

au prototipurile prezentate mai jos. Afișarea textelor presupune definirea unor parametri care pot fi gestionați prin aceste funcții. Valorile definite sunt, de asemenea, prezentate, după cum urmează:

<p>void settextstyle(int <i>font</i>, int <i>direction</i>, int <i>charsize</i>);</p> <ul style="list-style-type: none"> ▪ <i>Font</i> 0 (DEFAULT_FONT), 1 (TRIPLEX_FONT), 2 (SMALL_FONT), 3 (SANS_SERIF_FONT), 4 (GOTHIC_FONT); ▪ <i>Direction</i> HORIZ_DIR (de la stânga la dreapta), VERT_DIR (de jos în sus); ▪ <i>Charsize</i> 1 (8*8 pixeli), 2 (16*16 pixeli) ... 10 (80*80 pixeli). Dimensiunea poate fi stabilită de utilizator folosind funcția "setusercharsize" (a se vedea mai jos). În acest caz parametrul "<i>charsize</i>" va avea valoarea zero. 	<p>– funcția setează setul de caractere folosit (definit prin parametrul <i>font</i>), direcția de scriere (<i>direction</i>) și dimensiunea caracterului în pixeli (<i>charsize</i>). Valorile predefinite pentru parametri sunt:</p>
<p>void settextjustify(int <i>horiz</i>, int <i>vert</i>);</p> <ul style="list-style-type: none"> ▪ <i>Horiz</i> LEFT_TEXT, CENTER_TEXT, RIGHT_TEXT (poziția curentă se află în stânga, în centru, respectiv, în dreapta); ▪ <i>Vert</i> BOTTOM_TEXT, CENTER_TEXT, TOP_TEXT (poziția curentă se află în marginea inferioară, centru, respectiv, în marginea superioară). 	<p>– funcția definește, față de poziția curentă, cadrajul textului pe orizontală (<i>horiz</i>) și, respectiv, pe verticală (<i>vert</i>).</p>
<p>void gettextsettings(struct textsettingstype *<i>textinfo</i>);</p>	<p>– funcția determină valorile curente pentru parametrii (de tip int) <i>font</i>, <i>direction</i>, <i>charsize</i>, <i>horiz</i> și <i>vert</i> – informații cu care umple structura textsettingstype către care poartă textinfo.</p>
<p>void setusercharsize(int <i>multx</i>, int <i>divx</i>, int <i>multy</i>, int <i>divy</i>);</p>	<p>– funcția permite definirea de către utilizator a dimensiunii caractere-lor, anume, acestea se definesc prin înmulțirea lățimii lor cu <i>multx/divx</i> și a înălțimii cu <i>multy/divy</i>; valorile astfel setate devin active <u>doar dacă valoarea lui <i>charsize</i> (setată printr-un apel prealabil al funcției <i>settextstyle</i>) este 0.</u></p>

După setarea parametrilor de mai sus se pot afișa texte folosind funcțiile "**outtext**" și "**outtextxy**". Parametrul *sir*, cu semnificație comună în cele două funcții, este un pointer către zona de memorie în care se păstrează textul de afișat. Funcțiile afișează caractere colorate folosind culoarea de desenare curentă. Dimensiunile în pixeli ale unui șir de caractere se pot determina folosind funcțiile "**textheight**" și "**textwidth**".

<p>void outtext(char *<i>sir</i>);</p>	<p>afișare text în poziția curentă a cursorului;</p>
<p>void outtextxy(int <i>x</i>, int <i>y</i>, char *<i>sir</i>);</p>	<p>afișare text începând din punctul de coordonate (<i>x,y</i>) al ecranului;</p>
<p>int textheight(char *<i>sir</i>);</p>	<p>funcția determină, pe baza mărimii curente a fontului și a factorului de multiplicare (setat cu <i>settextstyle</i>) mărimea în pixeli a înălțimii șirului – valoare pe care o și returnează.</p>

```
int textwidth(char *sir);
```

funcția determină, pe baza lungimii șirului, a mărimii curente a fontului și a factorului de multiplicare mărimea în pixeli a lățimii șirului – valoare pe care o și returnează.

5. Gestiunea imaginilor

Biblioteca grafică conține câteva funcții ce țin de lucrul la nivel de pixel, de gestiunea imaginilor și, respectiv, de gestiunea unor zone din ecran; dintre acestea amintim:

Lucrul la nivel de pixel

<code>void putpixel(int <i>x</i>, int <i>y</i>, int <i>culoare</i>);</code>	– afișează pe ecran un pixel, în punctul (<i>x,y</i>) și în culoarea dată de parametrul <i>culoare</i> ⁶ ;
<code>unsigned getpixel(int <i>x</i>, int <i>y</i>);</code>	– determină culoarea pixelului din poziția (<i>x,y</i>) a ferestrei grafice;
<code>void moveto(int <i>x</i>, int <i>y</i>);</code>	– definește poziția pixelului curent prin mutarea poziției curente în punctul (<i>x,y</i>);
<code>void moverel(int <i>dx</i>, int <i>dy</i>);</code>	– mută poziția curentă <i>dx</i> pixeli pe direcția <i>x</i> și <i>dy</i> pixeli pe direcția <i>y</i> ⁷ .

Gestiunea imaginilor

<code>void getimage(int <i>st</i>, int <i>sus</i>, int <i>dr</i>, int <i>jos</i>, void *<i>bitmap</i>);</code>	copie imaginea unei zone dreptunghiulare de pe ecran în memorie (la adresa <i>bitmap</i>); primele două cuvinte din zona de memorie păstrează informația de lățime, respectiv, înălțime ale dreptunghiului copiat;
<code>unsigned imageSize(int <i>st</i>, int <i>sus</i>, int <i>dr</i>, int <i>jos</i>);</code>	determină mărimea (în bytes) a spațiului de memorie necesară pentru a stoca o imagine; dacă această mărime $\geq 64K-1$ bytes, funcția returnează 0xFFFF(-1). Funcția se apelează înaintea funcției <i>getimage</i> , fiind utilă în alocarea spațiului necesar de memorie (alocare realizată uzual în memoria heap).
<code>void putimage(int <i>st</i>, int <i>sus</i>, void *<i>bitmap</i>, int <i>op</i>);</code>	realizează operația inversă funcției <i>getimage</i> (de afișare a imaginii de la adresa <i>bitmap</i>); parametrul <i>op</i> controlează modul cum se calculează culoarea pentru fiecare pixel-destinație de pe ecran; pentru aceasta se pornește de la pixelul deja existent pe ecran și, respectiv, pixelul sursă corespondent, existent în memorie. Astfel, pentru <i>op</i> avem:
valorile:	0 (COPY_PUT) copie imaginea ca atare; 1 (XOR_PUT) implementează operația <i>sau exclusiv</i> între imaginea sursă și imaginea deja existentă pe ecran; 2 (OR_PUT) implementează operația <i>sau</i> între aceleași date menționate; 3 (AND_PUT) implementează operația <i>și logic</i> iar opțiunea 4 (NOT_PUT) copiază imaginea din memorie pe ecran complementând datele aflate în memorie.

Gestiunea unor zone din ecran

Ecranul poate fi partajat în mai multe părți ce pot fi gestionate independent. Aceste părți le vom numi, în continuare, tot ferestre grafice.

<code>void setviewport(int <i>st</i>, int <i>sus</i>, int <i>dr</i>, int <i>jos</i>, int <i>clip</i>);</code>	asigură partajarea ecranului în ferestre grafice. Apelul funcției stabilește o nouă
--	---

⁶ Reamintim aici faptul că parametrul *culoare* reprezintă indicele culorii de desenare curentă așa cum apare ea în tabela ce definește paleta curentă.

⁷ Majoritatea funcțiilor ce au ca parametri coordonate de pixel, interpretează aceste coordonate ca fiind relative față de fereastra activă al cărei colț stânga sus are coordonatele (0,0).

	fereastră pentru ieșirea grafică, aceasta devenind chiar fereastra activă ⁸ . Colțurile noii ferestre sunt date de coordonatele (<i>st, sus</i>) și (<i>dr, jos</i>) iar în noua fereastră poziția curentă (CP) este mutată în punctul (0,0). Parametrul <i>clip</i> determină, atunci când are valori diferite de zero, trunchierea desenelor în limitele ferestrei active.
<code>void clearviewport(void);</code>	curăță fereastra activă și mută CP în poziția (0,0) a acesteia; după apelul funcției toți pixelii ferestrei au aceeași culoare (culoarea de fond curentă).
<code>void cleardevice(void);</code>	șterge tot ecranul (fereastra grafică implicită) iar CP este mutat în poziția de coordonate (0,0);
<code>void getviewsettings(struct viewporttype *fereastră);</code>	determină parametrii ferestrei active (<i>st, sus, dr, jos, clip</i>);

Obs.: Majoritatea funcțiilor ce au ca parametri coordonate de pixeli (de ex., funcțiile *outtextxy*, *putpixel*, *getpixel*, *moveto*, *getx*, *gety* etc.) au ca parametri coordonate relative la pixelul din colțul stânga sus al ferestrei active.

Memoria video necesară pentru a păstra starea ecranului setat în mod grafic se numește *pagină video*. Anumite combinații de moduri și hardware-uri video pot suporta mai multe pagini video. Astfel, în cazul adaptoarelor de tip EGA/VGA, memoria video este capabilă să memoreze 8 pagini, numerotate de la 0 la 7. Câteva funcții de lucru cu aceste pagini sunt redate mai jos și ele sunt folosite pentru animație (aceasta din urmă se obține prin alternarea paginii vizuale). Facem remarcă că, funcțiile de desenare și scriere de texte acționează asupra unei singure pagini, numită și *pagină activă*. De regulă pagina activă este vizualizată pe ecran. Programatorul are, însă, posibilitatea să vizualizeze și o altă pagină decât cea activă. Pagina grafică vizuală (pagina din memorie de unde se va prelua informația ce se va afișa pe ecran) poate să nu fie aceeași cu pagina activă (pagina unde au loc operațiile de desenare curente). Într-un astfel de caz pagina grafică activă poate fi construită fără a afecta imaginea afișată pe ecran.

<code>void setactivepage(int nrpag);</code>	– face pagina <i>nrpag</i> pagina grafică activă (în care au loc toate operațiile grafice);
<code>void setvisualpage(int nrpag);</code>	– face pagina <i>nrpag</i> pagina grafică vizuală (pagina vizibilă pe ecran);
<code>int getactivepage(void);</code> ^{win}	– determină numărul paginii active curente;
<code>int getvisualpage(void);</code> ^{win}	– determină numărul paginii vizuale curente.
<code>void swapbuffers(void);</code> ^{win}	– interschimbă rolurile buffer-elor grafice vizual și activ curente;

Funcția *swapbuffers* este echivalentă cu următoarea secvență de cod:

```

int oldv = getvisualpage( );
int olda = getactivepage( );
setvisualpage(olda);
setactivepage(oldv);

```

6. Desenare și colorare

Forme geometrice unidimensionale (segmentul de dreaptă)

<code>void line(int x₁, int y₁, int x₂, int y₂);</code>	– trasează un segment de dreaptă între punctele (<i>x₁, y₁</i>) și (<i>x₂, y₂</i>); culoarea, stilul de linie și
---	--

⁸ Inițial (imediat după setarea modului grafic), fereastra activă o reprezintă tot ecranul (fereastra implicită).

void lineto (int x, int y);	<p>grosimea sunt cele curente (setate anterior pentru desenare); nu se actualizează CP;</p> <ul style="list-style-type: none"> - trasează un segment de dreaptă între poziția curentă și punctul (x,y) după care mută CP în punctul (x,y);
void linerel (int dx, int dy);	<ul style="list-style-type: none"> - trasează un segment între punctele (x₀,y₀) și (x₀+dx,y₀+dy), unde x₀ și y₀ sunt coordonatele poziției curente; CP este mutat apoi în punctul final al segmentului;
Forme geometrice bidimensionale (cercul, elipsa, dreptunghiul, poligonul etc.)	
void circle (int x _c , int y _c , int r);	<ul style="list-style-type: none"> - desenează un cerc centrat în (x_c,y_c) și de rază r.
void arc (int x _c , int y _c , int <i>unghistart</i> , int <i>unghifin</i> , int r);	<ul style="list-style-type: none"> - desenează un arc de cerc centrat în (x_c,y_c) și de rază r. Arcul de cerc merge de la <i>unghistart</i> la <i>unghifin</i> (similar arcului trigonometric: 0° → ora 3, 90° → ora 12 etc.).
void ellipse (int x _c , int y _c , int <i>unghistart</i> , int <i>unghifin</i> , int <i>semiamaxamare</i> , int <i>semiamamica</i>);	<ul style="list-style-type: none"> - desenează un arc de elipsă centrată în (x_c,y_c) și având razele orizontală și, respectiv, verticală date de <i>semiamaxamare</i> și <i>semiamamica</i>. Arcul de cerc merge de la <i>unghistart</i> la <i>unghifin</i> (similar arcului trigonometric); pentru valorile particulare (0°, 360°) pentru aceste unghiuri se obține o elipsă completă.
void rectangle (int st, int sus, int dr, int jos);	<ul style="list-style-type: none"> - desenează un dreptunghi având colțurile stânga-sus și dreapta-jos date de coordonatele (st,sus) și (dr,jos).
void drawpoly (int nrpct, int *polypct);	<ul style="list-style-type: none"> - trasează o linie poligonală determinată de nrpct puncte ale căror coordonate formează secvența de întregi memorată în vectorul polypct; funcția nu închide automat linia poligonală, aceasta putând fi realizată numai atunci când se specifică pentru ultimul punct aceleași coordonate ca și cele aferente punctului de început al liniei poligoale.
Umplerea/colorarea formelor geometrice bidimensionale	
void fillellipse (int x _c , int y _c , int <i>semiamaxamare</i> , int <i>semiamamica</i>);	<ul style="list-style-type: none"> - desenează o elipsă cu centrul în punctul (x_c,y_c) și având razele orizontală și, respectiv, verticală date de <i>semiamaxamare</i> și <i>semiamamica</i>, elipsă pe care o umple cu culoarea și pattern-ul curente.
void bar (int st, int sus, int dr, int jos);	<ul style="list-style-type: none"> - desenează un domeniu dreptunghiular colorat, folosind culoarea și pattern-ul de umplere curente; funcția nu realizează și conturul acestuia, scop pentru care se poate folosi funcția <i>bar3d</i>, cu parametrul de <i>adâncime</i> setat la valoarea 0 sau funcția <i>rectangle</i>.
void bar3d (int st, int sus, int dr, int jos, int <i>adâncime</i> , int <i>topflag</i>);	<ul style="list-style-type: none"> - pentru valori <i>topflag</i> diferite de zero funcția desenează o prismă dreptunghiulară dreaptă și umple cu pattern-ul și culoarea de umplere curente această prismă având adâncimea dată de parametrul <i>adâncime</i>. Parametrul <i>topflag</i> setează trasarea (pentru valori nenule) sau nu (pentru valoarea zero) a feței superioare a prisme.
void fillpoly (int nrpct, int *polypct);	<ul style="list-style-type: none"> - desenează conturul unui poligon dat de nrpct puncte ale căror coordonate formează secvența de întregi memorată în vectorul polypct; stilul liniei și culoarea sunt cele curente; în plus, funcția colorează poligonul folosind pentru umplere pattern-ul și culoarea de umplere curente (setate anterior cu <i>setfillstyle</i>); spre deosebire de <i>drawpoly</i>, <i>fillpoly</i> închide automat poligonul.

void pieslice (int x_c , int y_c , int <i>unghistart</i> , int <i>unghifin</i> , int <i>raza</i>);	- desenează și umple un sector de cerc centrat în (x_c, y_c) și de rază dată, <i>raza</i> . Arcul de cerc pornește de la <i>unghistart</i> și se termină la <i>unghifin</i> .
void sector (int x_c , int y_c , int <i>unghistart</i> , int <i>unghifin</i> , int <i>semiaxamare</i> , int <i>semiaxamica</i>);	- desenează și umple un sector dintr-o elipsă centrată în punctul (x_c, y_c) , având razele orizontală și, respectiv, verticală date de <i>semiaxamare</i> și <i>semiaxamica</i> ; arcul de cerc merge de la <i>unghistart</i> la <i>unghifin</i> (similar parcurgerii arcului trigonometric).
void floodfill (int x , int y , int <i>culoare</i>)	- colorează un domeniu închis obținut nu folosind strict o funcție standard din cele prezentate ci folosind două sau mai multe astfel de funcții implicate în trasarea elementelor componente ale figurii (de ex., <i>line</i> , <i>lineto</i> , <i>arc</i> etc.). Parametrii (x, y) sunt coordonatele unui punct interior domeniului ce se colorează iar <i>culoare</i> definește culoarea folosită în desenarea conturului figurii.

Ca regulă generală: conturul figurilor geometrice se realizează în culoarea de desenare curentă (setată cu funcția *setcolor*), în timp ce colorarea acestora se realizează în culoarea de umplere curentă, respectiv, folosind o colorare cu culoare de fond, o colorare uniformă sau o colorare prin hașură (culoarea și pattern-ul de umplere sunt setate cu funcția *setfillstyle*).

Întrucât pixelii nu sunt puncte ideale (având forme dreptunghiulare), figurile afișate pe ecran apar adesea ca fiind deformate. Pentru a corecta acest lucru se folosesc doi factori de corecție relativ la abscisă, respectiv, ordonată setați prin funcția *setaspectratio*.

void setaspectratio (int x_{aspect} , int y_{aspect});	- coeficienții de aspect (x_{aspect}, y_{aspect}) , depind de adaptorul grafic.
void getaspectratio (int * x_{aspect} , int * y_{aspect})	- determină și memorează, în zonele spre care pointează parametrii x_{aspect} și y_{aspect} , cei doi coeficienți de aspect curenți.

5. Gestiunea lucrului cu mouse-ul

Dintre funcțiile *winbgi* dedicate lucrului cu mouse-ul amintim următoarele:

int mousex (void); int mousey (void);	returnează coordonatele curente x și, respectiv, y ale mouse-ului; în cazul în care mouse-ul nu a fost niciodată în fereastra grafică, ambele coordonate returnate vor fi 0; în cazul poziționării mouse-ului în afara ferestrei grafice, se păstrează ultimele coordonate cunoscute.
bool ismouseclick (int <i>kind</i>);	returnează valoarea adevărat (1) dacă există un eveniment de mouse neprocesat, eveniment de tipul <i>kind</i> specificat (constantele posibile pentru parametrul <i>kind</i> sunt prezentate în Anexa 2(6)).

Un eveniment de mouse poate fi procesat prin apelul uneia din următoarele două funcții:

void getmouseclick (int <i>kind</i> , int& x , int& y);	salvează în x , y coordonatele unui eveniment neprocesat de tipul specificat (<i>kind</i>); în absența unui astfel de eveniment funcția returnează -1 pentru x și y . După apelul funcției, <i>ismouseclick</i> va returna valoarea de adevăr fals (0) până când un nou eveniment de tipul <i>kind</i> va avea loc.
void clearmouseclick (int <i>kind</i>);	funcție similară lui <i>getmouseclick</i> cu deosebirea că nu furnizează coordonatele evenimentului.

Exemplificări simple ale comenzilor grafice discutate mai sus, respectiv, exemple de probleme folosind aceste comenzi, le puteți găsi în directoarele GRAFICA_exemple, respectiv, GRAFICA_probleme.