Moduri de lucru cu monitoarele video ale calculatoarelor personale

Scopul lucrării

a) Studiul modului de lucru text.
b) Studiul modului de lucru grafic.
c) Utilizarea funcțiilor de nivel înalt pentru controlul modurilor de lucru ale monitoarelor video.

6.1. Introducere

Pentru un calculator personal (PC) există două moduri diferite de afişare a informațiilor pe un *ecran*: mod *text* (caractere alfabetice, numerice, de punctuație şi speciale) şi mod *grafic*. Toate dispozitivele de afişare, numite *monitoare* sau *display*-uri utilizează tehnica de compunere a imaginilor text sau grafice prin "aprinderea" pe ecran cu intensități şi culori diferite a unor zone de dimensiuni foarte reduse, aproape punctiforme, numite *pixeli*, organizate într-o rețea de puncte.

Densitatea acestei rețele de puncte ale ecranului, numită *rezoluție*, constituie o caracteristică importantă a echipamentului, considerat cu atât mai bun cu cât rezoluția este mai mare. Alte caracteristici care le diferențiază sunt: numărul de culori, viteza de afişare, numărul de puncte alocate zonei de afişare a unui caracter, etc. Interfațarea monitoarelor cu unitatea centrală este realizată prin intermediul unor dispozitive numite *adaptoare*, cum ar fi: CGA (Color Grafics Adapter), EGA (Extended Grafics Adapter), VGA (Video Grafics Adapter).

6.2. Modul grafic

În mod grafic, monitorul unui PC lucrează ca un televizor, afişând o imagine formată dintr-un număr mare de puncte independente, numite elemente de imagine sau *pixeli*. Culoarea şi luminozitatea fiecărui pixel pot fi stabilite independent de culoarea şi luminozitatea celorlalți pixeli.

Imaginea afişată de monitorul video este formată dintr-un număr oarecare de linii orizontale, fiecare linie având un număr oarecare de pixeli. Produsul acestora reprezintă numărul total de pixeli sau *rezoluția* ecranului. *Rezoluțiile tipice* ale ecranelor video în mod grafic sunt:

adaptor (*driver*)	moduri grafice (*graphics_modes*)	cod	coloane x linii	paletă culori	pagini
CGA	CGAC0	0	320 x 200	C0	1
	CGAC1	1	320 x 200	C1	1
	CGAC2	2	320 x 200	C2	1
	CGAC3	3	320 x 200	C3	1
	CGAHI	4	640 x 200	2 culori	1
MCGA	MCGAC0	0	320 x 200	C0	1
	MCGAC1	1	320 x 200	C1	1
	MCGAC2	2	320 x 200	C2	1

	MCGAC3	3	320 x 200	C3	1
	MCGAMED	4	640 x 200	2 culori	1
	MCGAHI	5	640 x 480	2 culori	1
EGA	EGALO	0	640 x 200	16 culori	4
	EGAHI	1	640 x 350	16 culori	2
EGA64	EGA64LO	0	640 x 200	16 culori	1
	EGA64HI	1	640 x 350	4 culori	1
EGA-MONO	EGAMONOHI	3	640 x 350	2 culori	1
	EGAMONOHI	3	640 x 350	2 culori	2
HERC	HERCMONOHI	0	720 x 348	2 culori	2
ATT400	ATT400C0	0	320 x 200	C0	1
	ATT400C1	1	320 x 200	C1	1
	ATT400C2	2	320 x 200	C2	1
	ATT400C3	3	320 x 200	C3	1
	ATT400MED	4	640 x 200	2 culori	1
	ATT400HI	5	640 x 400	2 culori	1
VGA	VGALO	0	640 x 200	16 culori	2
	VGAMED	1	640 x 350	16 culori	2
	VGAHI	2	640 x 480	16 culori	1
PC3270	PC3270HI	0	720 x 350	2 culori	1
IBM8514	IBM8514HI	1	1024 x 760	256 culori	
	IBM8514LO	0	640 x 480	256 culori	

6.3. Modul text

În mod text, caracterele sunt afişate pe *linii*, de la stânga la dreapta ecranului şi de sus în jos. Poziția curentă de afişare este indicată de un *cursor*, care se mută automat, odată cu afişarea caracterelor. Când cursorul atinge ultima poziție a ecranului (ultimul caracter al ultimului rând) întregul text afişat pe ecran se mută în sus cu o poziție, pierzându-se textul aflat pe prima linie şi creîndu-se spațiu pentru o linie nouă, la baza ecranului. Cursorul este poziționat în stânga noii linii.

În mod text, pentru fiecare caracter se păstrează în memorie două informații: *codul caracterului* care se afişează, şi *culorile* utilizate pentru desenarea *caracterului* şi respectiv a *fundalului* pe care se face afişarea. Structura octetului care codifică culorile de afişare este:

B	**F**	**F**	**F**	**C**	**C**	**C**	**C**
7	6	5	4	3	2	1	0

unde B = mod afişare (0 = continuă, 1 = intermitentă);
FFF = codul culorii utilizate pentru fundal;
CCCC = codul culorii utilizate pentru afişarea caracterului.

În afară de caracterele ASCII "obişnuite": litere, cifre, semne de punctuație, pentru afişarea pe ecran se pot utiliza şi caractere speciale din setul ASCII extins. Caracterele care fac parte din acest set şi nu fac parte din cel "clasic" sunt cele *semigrafice*, cu care se pot face diferite desene, caractere din alfabetul grecesc şi caractere cu specific matematic.

Pentru a indica poziția curentă, pe ecran se afişează un *cursor* de formă dreptunghiulară, a cărui dimensiune în cadrul unui caracter poate fi controlată prin program.

Pentru a memora conţinutul ecranului se utilizează o zonă de *memorie* specială în afara spaţiului de memorie utilizat pentru programe, denumită *memorie ecran*. Numărul maxim de puncte ce pot fi afişate şi numărul de culori utilizate pentru aceasta, în modul grafic determină dimensiunea memoriei ecran. Pentru o interfaţă clasică memoria utilizată pentru ecran are adresa de segment *0xB800*.

Deoarece pe ecran exstă mult mai multe puncte decât caractere, în memoria ecran pot fi memorate informaţiile corespunzătoare mai multor imagini de ecran text. De exemplu, pentru un ecran text în rezoluţie mare sunt necesari 80 x 25 x 2 = 4000 octeţi. În cazul unei interfeţe standard, memoria ecran are 16 kocteţi, în această memorie se pot păstra informaţiile pentru patru ecrane de tip text. Dintre aceste ecrane la un moment dat este activ (selectat) unul singur.

Adresa ocupată în memoria ecran de către informaţia corespunzătoare unui caracter este:

adr_car = (linie * 80 + coloana) * 2 + nr_ecran * 4096

Rezoluţiile tipice ale ecranelor video în mod text sunt:

- CGA: 25 de rânduri a 80 sau 40 de coloane de caractere în matrici de 8x8 pixeli, 16 culori;
- EGA: 25 sau 43 de rânduri a 80 de coloane de caractere în matrici de 7x16 pixeli, 16 culori;
- VGA: 25 sau 50 de rânduri a 80 de coloane de caractere în matrici de 9x16 puncte, 16 culori.

6.4. Exemple de funcţii C pentru controlul monitorului video

6.4.1. Configurarea sistemului grafic

detectgraph : Determinarea driver-ului şi a modului grafic utilizate verificând hardware-ul

Declaraţie: void far detectgraph(int far *graphdriver, int far *graphmode);

Parametru	*Semnificatie*
*graphdriver	intreg care specifică driver-ul grafic utilizat (constantă din enum. *graphics_drivers*)
*graphmode	specifică modul grafic initial (valoare din enum *graphics_modes*)

Observaţie: *detectgraph* detectează adaptorul grafic al sistemului şi alege modul care oferă cea mai mare rezolutie pentru adaptor.

initgraph : iniţializează sistemul grafic

Declaraţie: void far initgraph(int far *graphdriver, int far *graphmode, char far *pathtodriver);

Parametru	*Semnificatie*
*graphdriver	intreg care specifică driver-ul grafic utilizat (constantă din enum. *graphics_drivers*)
*graphmode	specifică modul grafic initial (valoare din enum *graphics_modes*)
pathtodriver	specifică calea către directorul în care *initgraph* caută driver-ele grafice (*.BGI) mai întâi (dacă nu sunt acolo, *initgraph* caută în directorul curent); dacă *pathtodriver* este nulă, fişierele driver trebuie să fie în directorul curent

Observaţii:

Pentru a utiliza sistemul grafic, trebuie mai întâi apelat *initgraph*.

initgraph inițializează sistemul grafic încărcând driver-ul grafic de pe disc (sau validând un driver înregistrat) trecând astfel sistemul în mod grafic.

initgraph resetează toate setările grafice (culoare, paletă, poziție curentă, etc.) la valoarea lor implicită şi resetează *graphresult* la 0. După apelul *initgraph*, **graphdriver* este setat la driver-ul grafic curent, iar **graphmode* este setat la modul grafic curent.

Se poate cere *initgraph* să utilizeze un driver şi mod grafic particular, sau să autodetecteze adaptorul video ataşat curent.

Dacă se cere *initgraph* să autodetecteze (**graphdriver = DETECT*), el apelează *detectgraph* pentru a selecta driver-ul şi modul grafic la cea mai mare rezoluție disponibilă pentru driver-ul detectat.

graphics_drivers : enumerare

constantă	valoare	constantă	valoare	constantă	valoare
DETECT	0 (cere autodetecție)	EGA64	4	ATT400	8
CGA	1	EGAMONO	5	VGA	9
MCGA	2	IBM8514	6	PC3270	10
EGA	3	HERCMONO	7		

closegraph : închide sistemul grafic

Declarație: void far closegraph(void);

Observație: *closegraph* eliberează toată memoria alocată pentru sistemul grafic, apoi reface ecranul în modul anterior apelului *initgraph.*

setviewport : Set-ează fereastra grafică curentă

Declaratie: void far setviewport(int left, int top, int right, int bottom, int clip);

Observații: (*left,top*) este colțul din stânga-sus, iar (*right,bottom*) este colțul din dreapta-jos al ferestrei. Poziția curentă (CP) este mutată în (0,0) al noii ferestre

setactivepage : Set-ează pagina activă
setvisualpage : Set-ează pagina vizibilă

Declarații: void far setactivepage(int page);
void far setvisualpage(int page);

getcolor : returnează culoarea curentă pentru desenare
setcolor : set-ează culoarea curentă pentru desenare

Declarații: int far getcolor(void);
void far setcolor(int color);

getbkcolor : returnează culoarea curentă pentru fond
setbkcolor : set-ează culoarea curentă pentru fond

Declarații: int far getbkcolor(void);
void far setbkcolor(int color);

CGA_COLORS : enumerare

Nume paletă	Constantă asociată 1	2	3
CGA0	CGA_LIGHTGREEN	CGA_LIGHTRED	CGA_YELLOW
CGA1	CGA_LIGHTCYAN	CGA_LIGHTMAGENTA	CGA_WHITE
CGA2	CGA_GREEN	CGA_RED	CGA_BROWN
CGA3	CGA_CYAN	CGA_MAGENTA	CGA_LIGHTGRAY

EGA_ COLORS : enumerare

Constantă	Valoare	Constantă	Valoare
EGA_BLACK	0	EGA_DARKGRAY	56
EGA_BLUE	1	EGA_LIGHTBLUE	57
EGA_GREEN	2	EGA_LIGHTGREEN	58
EGA_CYAN	3	EGA_LIGHTCYAN	59
EGA_RED	4	EGA_LIGHTRED	60
EGA_MAGENTA	5	EGA_LIGHTMAGENTA	61
EGA_LIGHTGRAY	7	EGA_YELLOW	62
EGA_BROWN	20	EGA_WHITE	63

settextstyle : set-ează caracteristicile curente ale textului

Declaratie: void far settextstyle(int font, int direction, int charsize);

6.4.2. Utilizarea sistemului grafic

cleardevice : şterge ecranul grafic

Declarație: void far cleardevice(void);

Observații: ştergerea constă în acoperirea cu culoarea curentă a fundalului
cleardevice şterge întregul ecranul grafic şi muta CP (poziția punctului curent) în origine (0,0).

imagesize : returnează numărul de octeți necesari pentru stocarea unei imagini

Declarație: unsigned far imagesize(int left, int top, int right, int bottom);

getimage : salvează o imagine dintr-o regiune specificată în memorie
putimage : pune o imagine pe ecran

Declarații: void far getimage(int left, int top, int right, int bottom, void far *bitmap);
void far putimage(int left, int top, void far *bitmap, int op);

Observații: *putimage* repune o imagine anterior salvată cu *getimage* pe ecran, cu colțul din stînga sus în (left,top).

getimage copiază o imagine de pe ecran în memorie.

Parametru	*Semnificatie*
bitmap	adresa unei zone din memorie în care este salvată imaginea; primele două cuvinte ale zonei sunt utilizate pentru lățimea şi înălțimea dreptunghiului
op	specifică un operator de combinare care controlează modul de calcul al culorii pentru fiecare pixel destinație pe ecran, bazîndu-se pe pixel-ul aflat deja pe ecran şi sursa corespunzătoare din memorie

putimage_ops: enumerare, dă numele operatorilor de combinare pentru *putimage*.

Constantă	Valoare	Funcție
COPY_PUT	0	Copiază imaginea sursă pe ecran
XOR_PUT	1	OR eXclusiv între imaginea sursă şi cea aflată deja pe ecran
OR_PUT	2	OR între imaginea sursă şi cea aflată deja pe ecran
AND_PUT	3	AND între imaginea sursă şi cea aflată deja pe ecran
NOT_PUT	4	Copiază inversul imaginii sursă pe ecran

getmaxx : returnează coordonata x (ordonata) maximă a ecranului
getmaxy : returnează coordonata y (abscisa) maximă a ecranului

Declarații: int far getmaxx(void);
int far getmaxy(void);

getx : returnează poziția curentă a coordonatei x
gety : returnează poziția curentă a coordonatei y

Declarații: int far getx(void);
int far gety(void);

moverel : mută poziția curentă (CP) la o distanță relativă
moveto : mută CP în (x, y)

Declarații: void far moverel(int dx, int dy);
void far moveto(int x, int y);

getpixel : obține culoarea unui pixel specificat (x,y)
putpixel : desenează un pixel într-un punct specificat (x,y)

Declarații: unsigned far getpixel(int x, int y);
void far putpixel(int x, int y, int color);

line : desenează o linie între două puncte specificate

linerel : desenează o linie la distanță relativă față de poziția curentă (CP)
lineto : desenează o linie din CP la (x,y)

Declarații: void far line(int x1, int y1, int x2, int y2);
void far linerel(int dx, int dy);
void far lineto(int x, int y);

Observații: *line* desenează o linie de la (x1, y1) la (x2, y2) utilizând culoarea, stilul liniei, şi grosimea curente fără a modifica poziția punctului curent (CP).

linerel desenează o linie de la CP la un punct care este la o distanță relativă (dx, dy) față de CP, apoi avansează CP cu (dx, dy).

lineto desenează o linie de la CP la (x, y), apoi mută CP în (x, y).

rectangle : desenează un dreptunghi (în mod grafic)

Declarație: void far rectangle(int left, int top, int right, int bottom);

Observații: *rectangle* desenează un dreptunghi cu culoarea, stilul liniei, şi grosimea curente.

(*left,top*) este colțul din stânga-sus al dreptunghiului, iar (*right,bottom*) este colțul din dreapta-jos al dreptunghiului.

arc : desenează un arc de cerc
circle : desenează un cerc
pieslice : desenează şi umple un sector de cerc

Declarații: void far arc(int x, int y, int stangle, int endangle, int radius);
void far circle(int x, int y, int radius);
void far pieslice(int x, int y, int stangle, int endangle, int radius);

Parametru	*Semnificatie*
(x,y)	Centrul cercului
stangle	Unghiul inițial în grade
endangle	Unghiul final în grade
radius	Raza cercului

outtext : afişează un şir de caractere într-o fereastră video
outtextxy : afişează un şir de caractere într-o locație specificată (x,y)

Declarații: void far outtext(char far *textstring);
void far outtextxy(int x, int y, char far *textstring);

6.4.3. Configurarea modului text

textattr : set-ează atributele textului pentru ferestre de text
textbackground : selectează o nouă culoare a fondului textului
textcolor : selectează o nouă culoare a caracterului

Declarații: void textattr(int newattr);
void textbackground(int newcolor);
void textcolor(int newcolor);

Parametru	*Semnificatie*
newattr	Informațiile culorilor codate
Exemplu:	textcolor(CYAN + BLINK);

COLORS: enumerare

Constantă	Valoare	Utilizată ca fond	Constantă	Valoare	Utilizată ca fond
BLACK	0	Da	DARKGRAY	8	Nu
BLUE	1	Da	LIGHTBLUE	9	Nu
GREEN	2	Da	LIGHTGREEN	10	Nu
CYAN	3	Da	LIGHTCYAN	11	Nu
RED	4	Da	LIGHTRED	12	Nu
MAGENTA	5	Da	LIGHTMAGENTA	13	Nu
BROWN	6	Da	YELLOW	14	Nu
LIGHTGRAY	7	Da	WHITE	15	Nu
BLINK	128	Nu ***			

*** Pentru a afişa caractere intermitent în mod text, trebuie adunat BLINK la culoarea fondului.

6.4.4. Utilizarea modului text

window : Defineşte fereastra activă în mod text

Declarație: void window(int left, int top, int right, int bottom);

wherex : returnează poziția curentă pe orizontală a cursorului în fereastra text curentă
wherey : returnează poziția curentă pe verticală a cursorului în fereastra text curentă

Declarații: int wherex(void);
int wherey(void);

gotoxy : Poziționează cursorul în fereastra text

Declarație: void gotoxy(int x, int y);

gettext : copiază un text de pe ecran în mod text în memorie
puttext : copiază un text din memorie pe ecran în mod text

Declarații: int gettext(int left, int top, int right, int bottom, void*destin);
int puttext(int left, int top, int right, int bottom, void*source);

Observații: *gettext* memorează conținutul ecranului din dreptunghiul definit de (left, top) şi (right, bottom) în zona de memorie *destin. *puttext* scrie conținutul zonei de memorie *source pe ecran.

gettextinfo : Obține informațiile modului text
Declarație: void gettextinfo(struct text_info *r);

```
 struct text_info {
   unsigned char winleft;          /* coordonata din stânga a ferestrei */
   unsigned char wintop;           /* coordonata de sus a ferestrei */
   unsigned char winright;         /* coordonata din dreapta a ferestrei */
   unsigned char winbottom;        /* coordonata de jos a ferestrei */
   unsigned char attribute;        /* atributele textului */
   unsigned char normattr;         /* atributele normale */
   unsigned char currmode;         /* modul video curent: BW40, BW80, C40, C80, or C4350 */
   unsigned char screenheight;     /* înălțimea textului */
   unsigned char screenwidth;      /* lățimea textului */
   unsigned char curx;             /* coordonata x în fereastra curentă */
   unsigned char cury;             /* coordonata z în fereastra curentă */          };
```

6.5. Exemple de programe

```
{DETECTGR.C - Detectarea drive-ului si modului grafic}

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* numele diferitelor placi video suportate */
char *dname[] = { "autodetectat", "CGA", "MCGA", "EGA", "64K EGA",
                    "EGA monocrom", "IBM 8514",    "Hercules monocrom",
                    "AT&T 6300 PC", "VGA",         "IBM 3270 PC"         };
void main(void)
{
   /* returneaza informatia privind hardware-ul detectat */
   int gdriver, gmode, errorcode;
  /* detecteaza hardware-ul grafic disponibil */
   detectgraph(&gdriver, &gmode);
   /* citeste rezultatul apelului detectgraph */
   errorcode = graphresult();
   if (errorcode != grOk)  /* a aparut o eroare */
   {
      printf("Eroare grafica : %s\n", grapherrormsg(errorcode));
      printf(":");
      getch();
      exit(1); /* terminat cu cod de eroare */
   }
   /* afiseaza informatia detectata */
   clrscr();
   printf("Aveti o placa video %s.\n", dname[gdriver]);
   printf("Apasati orice tasta pentru oprire:");
   getch();
}


{GETIMAGE.C - Salveaza si reface continutul ecranului grafic}

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
int maxx, maxy;
```

```
void save_screen(void far *buf[4])
{
   unsigned size;
   int ystart=0, yend, yincr, block;
   yincr = (maxy+1) / 4;
   yend = yincr;
   size = imagesize(0, ystart, maxx, yend);
/* obtine dimensiunea in octeti a imaginii */
   for (block=0; block<=3; block++)
   {
      if ((buf[block] = farmalloc(size)) == NULL)
      {
         closegraph();
        printf("Eroare: nu este destul spatiu heap.\n");
         exit(1);
      }
      getimage(0, ystart, maxx, yend, buf[block]);
      ystart = yend + 1;
      yend += yincr + 1;
   }
}
void restore_screen(void far *buf[4])
{
   int ystart=0, yend, yincr, block;
   yincr = (maxy+1) / 4;
   yend = yincr;
   for (block=0; block<=3; block++)
   {
      putimage(0, ystart, buf[block], COPY_PUT);
      farfree(buf[block]);
      ystart = yend + 1;
      yend += yincr + 1;
   }
}
void main(void)
{
   int gdriver=DETECT, gmode, errorcode;
   void far *ptr[4];
   initgraph(&gdriver, &gmode, "c:\\borlandc\\bgi");
   errorcode = graphresult();
   if (errorcode != grOk)
   {
      printf("Eroare grafica: %s\n", grapherrormsg(errorcode));
      printf("Apasati orice tasta pentru oprire:");
      getch();
      exit(1);
   }
   maxx = getmaxx();
   maxy = getmaxy();
   /* deseneaza o imagine pe ecran */
   rectangle(0, 0, maxx, maxy);
   line(0, 0, maxx, maxy);
   line(0, maxy, maxx, 0);
   save_screen(ptr);    /* salveaza ecranul curent */
   getch();             /* pauza */
   cleardevice();       /* clear screen */
   getch();             /* pauza */
   restore_screen(ptr); /* restore the screen */
   getch();             /* pauza */
   closegraph();
}


{IMAGSIZE.C - Salveaza si reface continutul ecranului grafic}

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#define ARROW_SIZE 10

void draw_arrow(int x, int y)
```

```
{
   /* deseneaza o sageata pe ecran */
   moveto(x, y);
   linerel(4*ARROW_SIZE, 0);
   linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
   linerel(0, 2*ARROW_SIZE);
   linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}
void main(void)
{
   int gdriver = DETECT, gmode, errorcode;
   void *arrow;
   int x, y, maxx;
   unsigned int size;

   initgraph(&gdriver, &gmode, "c:\\bc31\\bgi");
   errorcode = graphresult();
   if (errorcode != grOk)  /* a aparut o eroare */
   {
      printf("Eroare grafica: %s\n", grapherrormsg(errorcode));
      printf("Apasa o tasta pentru oprire:");
      getch();
      exit(1); /* terminare cu un cod de eroare */
   }
   maxx = getmaxx();
   x = 0;
   y = getmaxy() / 2;
   /* deseneaza sageata */
   draw_arrow(x, y);
   /* calculeaza dimensiunea imaginii */
   size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE);
   /* aloca memorie pentru a stoca imaginea */
   arrow = malloc(size);
   /* copiaza sageata */
   getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE, arrow);
   /* repeta pana cand o tasta e apasata */
   while (!kbhit())
   {
      /* sterge vechea sageata */
      delay(10);
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
      x += ARROW_SIZE;
      if (x >= maxx)
         x = 0;
      /* afiseaza noua sageata */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
   }
   /* elibereaza memoria */
   free(arrow);
   closegraph();
}


{SETPAGE.C - Schimba pagina grafica activa si pagina grafica vizibila}

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

void main(void)
{
  /* selecteaza un driver si un mode care suporta pagini multiple */
  int gdriver = EGA, gmode = EGAHI, errorcode;
  int x, y, ht;

  initgraph(&gdriver, &gmode, "c:\\bc31\\bgi\\");
  errorcode = graphresult();
  if (errorcode != grOk)
    {
    printf("Eroare grafica: %s\n", grapherrormsg(errorcode));
    printf("Apasa orice tasta pentru oprire:");
```

```
    getch();
    exit(1);
  }
  x = getmaxx() / 2;
  y = getmaxy() / 2;
  ht = textheight("W");

  /*  selecteaza pagina invizibila pentru desenare (nr. 1) */
  setactivepage(1);
  /* deseneaza o linie in pagina nr. 1 */
  line(0, 0, getmaxx(), getmaxy());
  /* scrie un mesaj in pagina nr. 1 */
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  outtextxy(x, y, "Aceasta este pagina nr. 1:");
  outtextxy(x, y+ht, "Apasa orice tasta pentru oprire:");

  /* selecteaza desenare in pagina nr. 0 */
  setactivepage(0);
  /* scrie un mesaj in pagina nr. 0 */
  outtextxy(x, y, "Aceasta este pagina nr. 0.");
  outtextxy(x, y+ht, "Apasa orice tasta pentru a vedea pagina nr. 1:");
  getch();

  /* selecteaza pagina nr. 1 ca pagina visibila */
  setvisualpage(1);
  getch();
  closegraph();
}

{TEXTATTR.C - Modifica atributele textului}

#include <conio.h>
void main(void)
{
   int i,j;
   for (i=0; i<8; i++)
   {
    for (j=0; j<8; j++)
    {
       textattr(i + (j << 4));
       clrscr();
       cprintf("Acesta este un test\r\n");
       getch();
    }
   }
   clrscr();
}

{TEXTINFO.C - Obtine informatii despre modul text curent}

#include <conio.h>

int main(void)
{
   struct text_info ti;
   gettextinfo(&ti);
   clrscr();
   cprintf("stanga         %2d\r\n",ti.winleft);
   cprintf("sus            %2d\r\n",ti.wintop);
   cprintf("dreapta        %2d\r\n",ti.winright);
   cprintf("jos            %2d\r\n",ti.winbottom);
   cprintf("atribut        %2d\r\n",ti.attribute);
   cprintf("mod curent     %2d\r\n",ti.currmode);
   cprintf("inaltime       %2d\r\n",ti.screenheight);
   cprintf("latime         %2d\r\n",ti.screenwidth);
   cprintf("x curent       %2d\r\n",ti.curx);
   cprintf("y curent       %2d\r\n",ti.cury);
   getch();
}

{WINDOWS0.C - Creeaza si utilizeaza o fereastra in mod text}
```

```
#include <conio.h>
int i,j,a,b,l,c;
char ch;
struct text_info ti;
void fereastra(int li, int co, int st, int su)
{
   // desenare chenar
   window(st,su,st+co,su+li);
   textcolor(BLACK);textbackground(WHITE);
   cprintf("É");
   for (i=0; i<co-1; i++) cprintf("Í");
   cprintf("»");
   for (j=0; j<li-2; j++)
       {
       cprintf("ş");
       for (i=0; i<co-1; i++) cprintf(" ");
       cprintf("ş");
       }
   cprintf("Č");
   for (i=0; i<co-1; i++) cprintf("Í");
   cprintf("Ľ");
   // stergere continut
   window(st+1,su+1,st+co-1,su+li-2);
   textcolor(WHITE);textbackground(BLACK);
   clrscr();
}
void main(void)
{
   // salvarea dimensiunilor ferestrei curente
   gettextinfo(&ti);
   // citirea parametrilor noii ferestre
   cprintf("Introduceti numarul de linii (min 3)   ");
   scanf("%d", &l); // nr. linii
   cprintf("Introduceti numarul de coloane (min 3) ");
   scanf("%d", &c); // nr. coloane
   cprintf("Introduceti coordonata x stanga        ");
   scanf("%d", &a); // coloana stanga
   cprintf("Introduceti coordonata y sus           ");
   scanf("%d", &b); // linie sus
   clrscr();
   fereastra(l,c,a,b);     // crearea noii ferestre
   while ((ch != '.'))
       {
       ch = getch();
       putch(ch);
       }
   delay(500);
   // refacerea ferestrei initiale
   window(ti.winleft,ti.wintop,ti.winright,ti.winbottom);
   clrscr();
}
```

6.6. Desfăşurarea lucrării

a) Să se modurile de lucru cu monitorul video.

b) Să se studieze exemplele de la punctul 6.4. Să se verifice corectitudinea funcționării programelor.

c) Să se studieze programul demostartiv BGIDEMO.C din kit-ul de instalare Borland C ++.