Elemente de grafică ştiinţifică

Introducere

Sistemele de calcul moderne pun la dispoziţia utilizatorului două categorii de dispozitive grafice:

- Dispozitive cu *reprezentare discretă* (prin puncte), cele mai utilizate fiind monitoarele şi imprimantele grafice;
- Dispozitive cu *reprezentare continuă*, numite generic *plotters*.

Pentru dispozitivele cu reprezentare discretă, elemental grafic este numit *pixel* (de la picture element). Imaginea obţinută pe monitoarele grafice rezultă din senzaţia vizuală de ansamblu creată de pixeli şi este de fapt, reprezentarea desfăşurată a unei zone de memorie, numită *memorie grafică*. Pentru monitoarele monocrom starea de aprins/stins a fiecărui pixel corespunde stării 1/0 a unui bit al memoriei grafice. În cazul monitoarelor color sunt necesare zone de memorie suplimentare pentru informaţia de culoare si luminozitate a fiecărui pixel.

O imagine se caracterizează prin *rezoluţia* ei, adică numărul maxim de pixeli reprezentabili şi este specificată sub forma *n x m*, ca produs dintre numărul maxim de pixeli pe orizontală şi pe verticală. Rezoluţia este o reflectare directă a capacităţii memoriei grafice.

Astfel, monitoarele cu standardul VGA oferă o rezoluţie de 640 x 480 pixeli în 16 culori, cele cu standardul Super VGA au o rezoluţie de cel puţin 800 x 600 pixeli în 256 culori iar monitoarele cele mai performante, au rezoluţii de până la 1600 x 1200 pixeli in cca. 4 miliarde de culori.

Reprezentarea grafică a funcţiilor este o modalitate de vizualizare a datelor ştiinţifice. Sunt puse în evidenţă, în continuare, principiile care stau la baza aplicaţiilor grafice pe calculator. Cele mai utilizate medii de programare, sub sistemul de operare Windows sunt : Borland C++ şi Microsoft Visual C++.

Funcţii grafice în Borland C++

Fişierul header GRAPHICS.H din Borland C++ conţine o bibliotecă de peste 80 de funţii grafice elementare ce utilizează interfaţa grafică BGI (Borland Graphics Interface). Orice aplicaţie are acces la aceste funcţii prin directiva

```
# include <graphics.h>
```

Pentru realizarea efectivă a unei reprezentări grafice pe ecranul monitorului, aplicația trebuie să mai aibă acces la driverele grafice (fişiere cu extensia .BGI): *att.bgi, egavga.bgi, ibm8514.bgi*, etc.
Acestea stabilesc protocolul de comunicare dintre program şi ecranul grafic şi la fişierele cu seturi de caractere (cu extensia .chr).

Biblioteca GRAPHICS.H conține funcții care realizează inițializarea modului grafic, selectarea culorilor, grosimea liniilor caracteristicile setului de caractere şi desenarea unor figuri standard: Punct, segment, arc de cerc, cerc, elipsă, arc de elipsă, dreptunghi, poligon.

În realizarea reprezentărilor grafice un rol esențial îl are *cursorul grafic* (*pointerul*), care este asemănător cursorului din modul text, cu deosebirea că este punctual ş invizibil.

Funcțiile de desenare produc deplasarea cursorului grafic pe ecran, analog deplasării vârfului unui creion pe hârtie.

Ecranul grafic este un dreptunghi cu dimensiuni bine stabilite pentru fiecare tip de display şi depinde de modul grafic implementat.

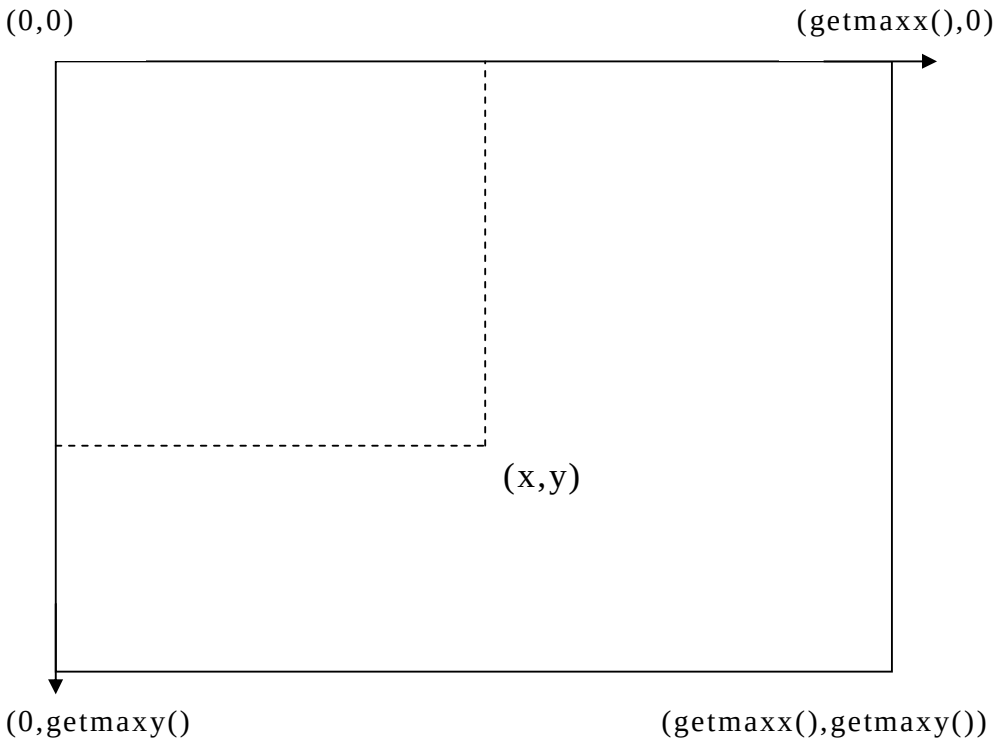

Fig. 1 Sistemul coordonatelor ecran al interfeței grafice BGI.

Sistemul de coordonate utilizat pentru precizarea poziției cursorului grafic are originea în colțul din stânga sus al ecranului grafic, ca în fig.1, iar coordonatele corespunzatoare *(x, y)* se numesc coordonate

ecran. Valoarea coordonatei ecran *x* (pe orizontală) creşte spre dreapta iar valoarea coordonatei ecran *y* (pe verticală) creşte în jos.

Valorile maxime ale coordonatelor *x* şi *y* corespunzătoare punctelor de pe ecran, depind de rezoluția imaginii; pentru ca un program să nu depindă de tipul de monitor, aceste valori maxime se obțin cu funcțiile *getmaxx()* şi *getmaxy().*

Cele mai utilizate funcții ale bibliotecii GRAPHICS.H sunt prezentate mai jos.

cleardevice()
Sintaxa: void far cleardevice(void);
Efect: şterge ecranul grafic, utilizând culoarea curentă a fondului şi poziționează cursorul grafic în origine (0, 0).

closegraph()
Sintaxa: void far closegraph(void);
Efect: încheie sesiunea de lucru în modul grafic şi restabileşte ecranul anterior; eliberează memorie alocată sistemului grafic.

getmaxx() şi getmaxy()
Sintaxa: int far getmaxx(void); int far getmaxy(void);
Efect: oferă la apelare, valorile legale maxime ale coordonatelor ecran x şi respectiv y. De exemplu, pentru un monitor VGA, cu rezoluția 640x480 pixeli, sunt returnate valorile 639 şi respectiv 479.

initgraph()
Sintaxa: void far initgraph(int far *graphdriver, int far *graphdriver, int far *graphmode);
Efect: inițializează modul grafic, utilizând parametrii *graphdriver, şi *graphmode, pentru specificarea driverului şi modului grafic şi parametrul de tip *string* *pathtodriver, pentru a se localiza directorul în care a fost instalat driverul grafic (.bgi) corespunzător.

Funcția initgraph() stabileşte valori implicite pentru toți parametrii grafici (poziție curentă, culori etc.) .

Dacă parametrul actual comunicat lui *graphdriver este inițializat cu constanta predefinită DETECT, selectarea modului grafic se realizează automat. Exemplu:

```
{
      int gdriver = DETECT, gmode;
      initgraph(&gdriver, &gmode, ”c:/borlandc/bgi”);
}
```

line()
Sintaxa: void far line(int x1, int y1, int x2, int y2);
Efect: trasează o linie între punctele de coordonate (x1, y1) şi (x2, y2) Lăsând poziţia anterioară a cursorului neschimbată. Sunt utilizate culoarea, stilul şi grosimea de linie, curente (cum au fost setate anterior).

lineto()
Sintaxa: void far lineto(int x, int y);
Efect: trasează o linie de la poziţia actuală a cursorului grafic până în punctul de coordonate (x, y), care devine noua poziţie curentă a cursorului.

moveto()
Sintaxa: void far moveto(int x, int y);
Efect: modifică (mută) poziţia curentă a cursorului grafic în punctul de coordonate (x, y), care devine noua poziţie curentă a cursorului.

rectangle()
Sintaxa: void far rectangle(int left, int top, int right, int bottom);
Efect: desenează un dreptunghi definit prin colţul stânga-sus (left, top) şi colţul dreapta-jos (right, bottom), utilizând culoarea curentă şi tipul de linie curent.

settextstyle()
Sintaxa: void far settextstyle(int font, int direction, int charsize);
Efect: stabileşte tipul, direcţia şi dimensiunea setului de caractere utilizat de funcţiile outtext şi outtextxy. Parametrul font poate lua una din valorile predefinite, **DEFAULT_FONT**, **TRIPLEX_FONT, SMALL_FONT, SANS_SERIF_FONT, GOTHIC_FONT** iar parametrul direction ia una din valorile predefinite **HORIZ_DIR, VERT_DIR**.
Parametrul charsize (1..10) specifică mimea caracterelor.

settextjustify()
Sintaxa: void far settextjustify(int horiz, int vert);
Efect: defineşte poziţionarea textului faţă de poziţia cursorului grafic pentru funcţiile outtext() şi outtextxy(). Parametrii horiz şi vert poate lua valorile predefinite: **LEFT_TEXT, CENTER_TEXT, RIGHT_TEXT,** şi respectiv, **BOTTOM_TEXT, CENTER_TEXT, TOP_TEXT.**

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

Efect: scrie textul *textstring la poziția (x, y) utilizând atributele stabilite cu ajutorul funcțiilor settextjustify() şi settextstyle().

outtext()
Sintaxa: void far outtext(char far *textstring);
Efect: scrie textul *textstring la poziția curentă a cursorului grafic utilizând atributele stabilite cu ajutorul funcțiilor settextjustify() şi settextstyle().

Reprezentarea grafică a funcțiilor de o variabilă

Fie funcția $f : [x_{min}, x_{max}] \rightarrow [y_{min}, y_{max}]$, dată sub formă tabelară pe o rețea de puncte x_i :

$$f(x_i) = y_i \ , \ i = 1,2, \ . \ . \ . \ ,n.$$

În general, reprezentarea pe ecranul grafic nu poate utiliza direct *coordonatele* (x_i, y_i), pe care le vom numi *coordonate utilizator*. Acestea trebuie transformate mai întâi în *coordonate ecran,* (X_i, Y_i) pentru a se încadra în domeniile admise de valori pentru argumentele funcțiilor grafice. Acest proces de conversie se numeşte *scalare*.

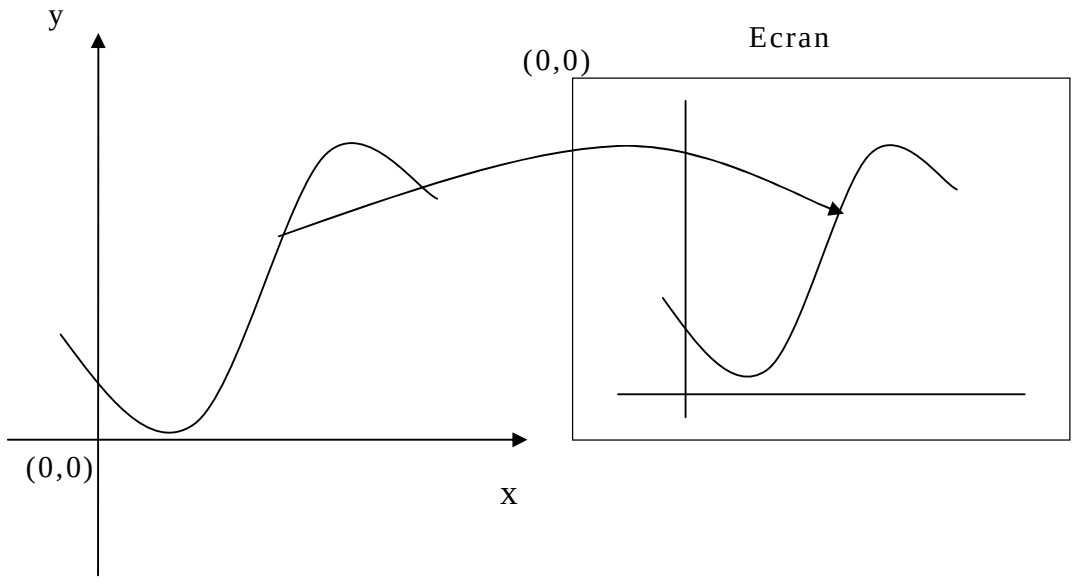

Fig. 2 Corespondența între coordonatele utilizator şi coordonatele ecran

Pentru ca în urma procesului de scalare, reprezentarea grafică pe ecran să rămână proporționată (figura să nu fie deformată), trebuie ca între coordonatele ecran şi coordonatele utilizator să existe o relație liniară:

$$X = A_x\, x + B_x$$
$$Y = A_y\, y + B_y$$

Coeficienții de scalare A_x , B_x se determină din condițiile:

$$X_{min} = A_x\, x_{min} + B_x$$
$$X_{max} = A_x\, x_{max} + B_x$$

de unde,

$$A_x = (X_{max} - X_{min})/(x_{max} - x_{min})$$
$$B_x = X_{min} - A_x x_{min}$$

Analog, coeficienții de scalare A_y , B_y se determină din relațiile:

$$A_y = (Y_{max} - Y_{min})/(y_{max} - y_{min})$$
$$B_y = Y_{min} - A_x y_{min}$$

La reprezentarea grafică a unei funcții de o variabilă, domeniul de definiție este cunoscut, deci se poate alege un interval închis de tip $[x_{min}, x_{max}]$, de trasare a graficului. Domeniul valorilor funcției, pentru $x \in [\, x_{min}, x_{max}]$, este tot interval şi anume: $[y_{min}, y_{max}]$, care inițial nu este cunoscut şi trebuie determinat.

Coordonatele ecran, corespunzătoare celor n puncte de tabelare, se determină aşadar din relațiile:

$$X_i = A_x\, x_i + B_x$$
$$Y_i = A_y\, y_i + B_y\ ,\quad i = 1, 2, 3, \ldots, n.$$

Algoritmul de calcul pentru reprezentarea grafică a unei funcții de o variabilă va fi:

1. Date de intrare:
   - expresia analitică a funcției de reprezentat, *f(x)*;
   - n = numărul de puncte utilizate pentru grafic;
   - intervalul de reprezentare: $[x_{min}, x_{max}]$,
   - X_{min} , X_{max} , Y_{min} , Y_{max} (care se calculează din cele precedente)

2. Inițializarea modului grafic;
3. Desenarea chenarului ferestrei de reprezentare;
4. Se atribuie valorile $x_{min}=x_1$, $x_{max}= x_n$;
5. Se calculează coeficienții de scalare pe direcția x:

$$A_x = (X_{max} - X_{min})/(x_{max} - x_{min})$$
$$B_x = X_{min} - A_x x_{min} ;$$

6. Se determină valorile y_{min}, y_{max} ;
7. Se calculează coeficienţii de scalare pe direcţia y :

$$A_y = (Y_{max} - Y_{min})/(y_{max} - y_{min})$$
$$B_y = Y_{min} - A_x y_{min} ;$$

8. Dacă $x_{min} \cdot x_{max} < 0$, se trasează axa ordonatelor la abscisa

$$X_0 = A_x \cdot 0 + B_x$$

adică între punctele (X_0, Y_{min}) şi (X_0, Y_{max}) ;

9. Dacă $y_{min} \cdot y_{max} < 0$, se trasează axa absciselor la ordonata

$$Y_0 = A_y \cdot 0 + B_y$$

adică între punctele (X_{min}, Y_0) şi (X_{max}, Y_0) ;

10. Poziţionează cursorul grafic în punctul (X_1, Y_1) ;

11. Desenează segmente între punctele (X_{i-1}, Y_{i-1}) şi (X_i, Y_i) unde

$$X_i = A_x x_i + B_x$$
$$Y_i = A_y y_i + B_y , \quad i = 1, 2, 3, \ldots, n.$$

Fişierul graphlib.h, care conţine toate funcţiile necesare pentru algoritmul de mai sus, poate fi inclus în programele ce necesită reprezentări grafice pentru funcţii de o variabilă.

Singura deosebire faţă de algoritmul de mai sus se referă la coordonatele ecran ce se transmit funcţiei Plot().

În loc să se transmită coordonatele ecran absolute (ixmin, iymin) şi (ixmax, iymax) ale colţurilor ferestrei de desenare, se transmit coordonatele relative:

fxmin = ixmin / getmaxx(), fymin = 1 – iymin / getmaxy()
fxmax = ixmax / getmaxx(), fymax = 1 – iymax / getmaxy()

În cazul axei y, se ţine cont de faptul că axa coordonatelor ecran şi cea a coordonatelor utilizator sunt orientate în sensuri opuse.

Pe baza coordonatelor ecran relative, rotunjite la partea întreagă, funcţia Plot() determină coordonatele absolute ale colţurilor ferestrei de reprezentare.

GRAFIC_FUNCTIE.CPP

```
 #include <stdio.h>
 #include <conio.h>
 #include <math.h>
 #include "memalloc.h"
 #include "graphlib.h"
 float Func(float x)
 {
 return (pow(x,3)-3*x*x+2)/(x*x+1.0);
}

 void main()
 {
   float h, xmin, xmax, *x, *y;
   int i,n;

   printf("xmin= "); scanf("%f", &xmin);
   printf("xmax= "); scanf("%f", &xmax);
   printf("n   = "); scanf("%i",&n);

  x = Vector(1,n);
  y = Vector(1,n);

   h = (xmax-xmin)/(n-1);
   for (i=1; i<=n; i++) {
   x[i] = xmin + (i-1)*h;
   y[i] = Func(x[i]);
   }

   InitGraph();
 //  Plot0(x,y,n,0.2, 0.8, 0.2, 0.8);
   Plot(x,y,n,1,0.1,0.95,0.1,0.95,"x","xx","DPLOT");
   getch();
   closegraph();
   FreeVector(x,1);
   FreeVector(y,1);
   }
```

GRAPHLIB>H

```
/* -------------------------graphlib.h----------------------*/
/* contine rutine pentru grafice de functii de o variabila,
 cu biblioteca grafica BGI din Borland C++---------------*/

 #ifndef GRAPHLIB
 #define GRAPHLIB

 #include <stdlib.h>
 #include <stdio.h>
```

```
 #include <string.h>
 #include <graphics.h>
 #include <math.h>
 #include <utils.h>
 int min(int a,int b)
 { return ((a<b)? a:b);}
 int max(int c, int d)
 {return ((c>d)? c:d);}
 void InitGraph(void)
 //initializeaza modul grafic
 {
  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "d:/borlandc/bgi");
 }
 /*================================================*/
 void Limits(float *xmin, float *xmax, float *scale, int *nsigd,
int *nintv)
 /*Pentru limitele xmax si xmin ale unui interval de valori
reale, returneaza  limitele intervalului extins care cuprinde un
numar intreg <10 de  subintervale cu lungimea exprimabila sub
forma d*10^p, unde d este 1,2 sau 5  iar p=puterea e un intreg;
       {
   return ( (value1 > value2) ? value1 : value2);
        }

 scale= factorul de scala 10^p
 nsigd= numarul relevant de cifre semnificative
 nintv= numarul de subintervale*/

 {
   const float eps=1e-3;    /*criteriul relativ de precizie*/
   const float xfact[3] = {0.5 ,0.5, 0.4};
   float corrmin, corrmax, factor, xmins, xmaxs, xnint;
   int i, modi;
   //se calculeaza factorul de scala initial:
   factor = ((*xmin)*(*xmax))?
1.0/(eps*min(Magn(*xmin),Magn(*xmax))):
                      1.0/(eps*max(Magn(*xmin),Magn(*xmax)));
   corrmin= 1.0 + Sign(eps,(*xmin));
   corrmax= 1.0 - Sign(eps,(*xmax)); /*corectii*/

   //inmultestefactor ciclic cu xfact[] pana cand nr de
subinterv xnint<10:
   for (i=1; i<=50; i++){
     xmins=floor((*xmin)*factor*corrmin);
     xmaxs=ceil ((*xmax)*factor*corrmax);
     xnint = abs(xmaxs - xmins);
     if (xnint<10) break;
     modi = i % 3;
     factor *= xfact[modi];
   }
```

```
   factor = (1.0 + eps)/factor;               /*corecteaza
factor*/
   *xmin = xmins*factor;
   *xmax = xmaxs*factor;
   *scale = max(Magn(*xmin),Magn(*xmax));      /*factorul de
scala*/
   factor = max(fabs(xmins), fabs(xmaxs));
   for (i=1; i<=modi; i++) factor /= xfact[i];
   *nsigd = log10(factor) +1;                   /*nr de cifre
semnificative*/
   *nintv = Nint(xnint);                        /*nr de
subintervale*/
 }


/*============================================================
*/
 void Format(float x,float scale, int nsigd, char mant[],char
expn[])
 /* Formateaza numarul x (cu factorul de scala) la nsigd cifre
semnific. returnand mantisa si exponentul lui 10 in mant[] si
expn[]*/
 {
   const int ndigmax = 5;    //numarul maxim de cifre
   int n, ndec, ndig;
   n = Nint(log10(scale));
   if ((n<0) || (n>3)) {
   sprintf(expn,"%i",n);
   x /= scale; n=0;
   }

   n++; //numarul de cifre inaintea punctului zecimal
   ndig = min(ndigmax, max(nsigd,n)); // numarul total de cifre
   ndec = ndig - n;                   // numarul de zecimale
   sprintf(mant,"%.*f", ndec,x);    // mantisa
  }
/*============================================================*/
  void Plot(float x[], float y[], int n, int style,
         float fxmin, float fxmax, float fymin, float fymax,
         char xtext[], char ytext[], char title[])
 /*  Realizeaza  reprezentarea  grafica  si  o  incadreaza  in
fereastra   [fxmin, fxmax] x [fymin, fymax], specificata prin
coordonate fractionare  din intervalul [0,1]. Limitele pe axele
x si y sunt extinse, pentru a  include un numar intreg <10 de
subintervale cu lungimea exprimabila sub  forma d*10^p. AXELE
SUNT INSCRIPTIONATE AUTOMAT.
 x[]= abscisele punctelor y[]= ordonatele punctelor
 n = numarul punctelor
 style = stilul desenului: =0 cu puncte, =1 cu linie continua
 fxmin = abscisa relativa minima a ferestrei 0<fxmin<fxmax<1
 fxmax = abscisa relativa maxima a ferestrei
```

```
 fymin = ordonata relativa minima a ferestrei 0<fymin<fymax<1
 fymax = ordonata relativa maxima a ferestrei
 xtext[]=textul axei x
 ytext[]=textul axei y
 title[]= titlul reprezentarii
 ============================================================*/
 {
 float ax, bx, ay, by, h, htic, scale, xmin, xmax, ymin, ymax;
 int ix, ixmin, ixmax, ixtext, iy, iymin, iymax, iytext;
 int charX, charY, i, nsigd, nintv, tic;
 char mant[5],expn[5], text[10];
 charX=textwidth ("0");
 charY=textheight("0"); //dimensiunea medie a caracterelor

 ixmin=Nint(fxmin*getmaxx()); iymin=Nint((1.0-ymin)*getmaxy());
 ixmax=Nint(fxmax*getmaxx()); iymax=Nint((1.0-fymax)*getmaxy());
 rectangle(ixmin,iymax,ixmax,iymin); // chenarul

 xmin=x[1]; xmax=x[n];               // axa x
 Limits(&xmin,&xmax,&scale, &nsigd, &nintv); // limitele extinse
 ax = (ixmax-ixmin)/(xmax-xmin);             // coeficienti de
scalare
 bx = ixmin - ax*xmin;
 h = (xmax-xmin)/nintv; htic=ax*h;
 tic = (ixmax-ixmin)/75;           // lungimea 'ticurilor'

 settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
 settextjustify(CENTER_TEXT, TOP_TEXT);
 iytext = iymax - 3*charY;
 outtextxy((ixmin+ixmax)/2,iytext,title);    // titlul
 iytext = iymin + charY;
  for (i=1; i<=nintv+1; i++) {
  ix=Nint(ixmin + (i-1)*htic);
  line(ix, iymin, ix, iymin-tic);
  line(ix, iymax, ix, iymax+tic);
  Format(xmin+(i-1)*h, scale, nsigd, mant, expn);
  outtextxy(ix-charX/2, iytext, mant);       // etichetele
  }

  iytext = iytext + 2*charY;
  strcpy(text, xtext);
  if ((scale<1.0) || (scale>1000.0)) strcat(strcat(text,"  *
1e"),expn);
  outtextxy((ixmin+ixmax)/2,iytext, text);     // eticheta axei

  ymin = y[1]; ymax = y[1];                    // axa Y
  for (i=2; i<=n; i++) {
      ymin =min(ymin, y[i]);
      ymax = max(ymax, y[i]);
  }
  if (ymin==ymax) { ymin *= 0.9; ymax *= 1.1;}
```

```
  Limits(&ymin, &ymax, &scale, &nsigd, &nintv);  // limitele
extinse
  ay = (iymax-iymin)/(ymax-ymin);
  by = iymin - ay * ymin;
  h = (ymax -ymin)/nintv; htic=ay*h;            // pas
inscriptionare
  settextjustify(RIGHT_TEXT, CENTER_TEXT);
  ixtext = ixmin - charY;

  for (i=1; i<=nintv+1; i++) {        // inscriptioneaza axa
  iy=Nint(iymin + (i-1)*htic);
  line(ixmin, iy, ixmin+tic, iy);
  line(ixmax, iy, ixmax-tic, iy);
  Format(ymin+(i-1)*h, scale, nsigd, mant, expn);
  outtextxy(ixtext, iy, mant);       // etichetele
  }

  ixtext= ixtext -charY - charX * strlen(mant);
  strcpy(text, ytext);
  if ((scale<1.0) || (scale>1000.0)) strcat(strcat(text,"  *
1e"),expn);
  settextstyle(DEFAULT_FONT,VERT_DIR,1);
  outtextxy(ixtext,(iymin+iymax)/2,text);   // eticheta axei

  if (xmin*xmax <0) line(Nint(bx), iymin, Nint(bx),iymax);
  if (ymin*ymax <0) line(ixmin, Nint(by), ixmax, Nint(by));
  ix=Nint(ax*x[1]+bx); iy=Nint(ay*y[1]+by);     // primul punct
  tic=(ixmax-ixmin)/125;
  if(style) moveto(ix,iy);
    else rectangle(ix-tic,iy-tic,ix+tic,iy+tic);

  for (i=2; i<=n; i++) {
  ix = Nint(ax*x[i]+bx); iy = Nint(ay*y[i]+by);
  if (style) lineto(ix,iy);
    else rectangle(ix-tic,iy-tic,ix+tic,iy+tic);
  }

 }
 #endif
```

MEMALLOC.H

```
#ifndef MEMALLOC
#define MEMALLOC
#include <stdio.h>
#include <stdlib.h>
  void FreeVector(float *p, int imin)
 {
  free((void*) (p+imin));
  }
float *Vector(int imin, int imax)
```

```
{
 float *p;

 p=(float*) malloc((size_t) ((imax-imin+1)*(sizeof(float))));
 if (!p) {
 printf("Vector: eroare de alocare!\n");
 exit(1);
 }
     return p - imin;
 }



  #endif
```

UTILS.H

```
 #ifndef UTILS
 #define UTILS
 #include <math.h>
 #define Sign(x,y) (y<0 ? -fabs(x): fabs(x)) //transfera lui x
semn y

 #define Nint(x) (int)floor(x+0.5)  // rotunjeste x la cel mai
aprop. intreg

 float Magn(float x) // return ordin de marime sub forma 10^n
 {
  if (x) return (fabs(x) >=1 ? pow(10, (int)(log10(fabs(x))))
                    :pow(0.1, (int)(fabs(log10(fabs(x))))+1));
                    else return 0.0;
                    }
 #endif
```

FRACTALI.CPP

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>

int L,grdriver,grmode,Eroare;
void Fractal(int x,int y,int R)
{ if (R>0) {
          Fractal(x-R, y+R,R/2);
           Fractal(x-R, y-R, R/2);
            Fractal(x+R, y-R, R/2);
             Fractal(x+R, y+R, R/2);
         bar(x-R/2,y-R/2,x+R/2,y+R/2);}
}
```

```
 int main()
 {
  cout<<"L= ";
 cin>>L;
grdriver=DETECT;
initgraph(&grdriver,&grmode, "D:\\BORLANDC\\BGI");
Eroare=graphresult();
     if (Eroare==grOk)
     { setbkcolor (WHITE);
     setfillstyle(SOLID_FILL,GREEN);
     Fractal(getmaxx()/2,getmaxy()/2,L);getch();}
closegraph();
     return 0  ;
  }
```