Janusz Sobieraj *Warszawa –sierpień 1994*

II.Niektóre zagadnienia grafiki w trybie 13h

W tym rozdziale zajmiemy się niektórymi problemami grafiki w trybie 320x200,256 kolorów,ze szczególnym uwzględnieniem tworzenia narzędzi graficznych bez korzystania z procedur i funkcji modułu *graph.tpu* .
Tryb ten można uruchomić na każdym komputerze wyposażonym w dowolną kartę VGA lub SVGA.Prosta organizacja tego trybu,łatwy dostęp do pamięci ekranu,pozwalają nawet niezawansowanemu użytkownikowi szybko uruchomić grafikę i wykonać pewne operacje graficzne.
Ważnym walorem tego trybu jest również możliwość operowania 256 kolorami,a po pewnych zmianach palety - nawet większą liczbą.
Rozwiązań, jakie zaproponujemy,nie należy uważać za ostateczne.Celowo posługujemy się prostym modelem,koncentrując się raczej na zasadach tworzenia podstawowych narzędzi. Wymagający użytkownik z łatwością stworzy bardziej wyrafinowane procedury, optymalizując zaproponowane rozwiązania.

II.1.Wykorzystanie modułu *GRAPH.*

Moduł *graph.tpu* pozwala,na komputerach wyposażonych w kartę VGA,uprawiać grafikę m.in. w trybach :
VGOLo =0 ,640x200 ,16 kolorów ,4 strony
VGAMed=1 ,640x350 ,16 kolorów ,2 strony
VGAHi =2 ,640x480 ,16 kolorów ,1 strona.
Obsługę tych trybów zapewnia sterownik *egavga.bgi*,do którego w czasie sesji graficznej kompilator powinien mieć wskazaną ścieżkę dostępu.Oto typowa procedura uruchamiająca tryb graficzny:

```
Program Rysuj;
uses graph;
var GrDriver,GrMode :integer;
      skala:real;
Procedure Grafika(tryb :integer);
 var s1,s2 :word;
  begin
    GrDriver:=9 ; GrMode:=tryb;
    InitGraph(GrDriver,GrMode,'');
    GetAspectRatio(s1,s2);
    skala:=s1/s2;
  end;
 ....
```

Zmienne globalne *GrDriver* i *GrMode* przechowują informację o sterowniku (w przykładzie ustalono VGA) oraz żądanym trybie pracy.Zmienna *skala* może służyć do skalowania niektórych operacji graficznych. Interesująca wydaję się być możliwość grafiki w trybie 320x200, 256 kolorów. W celu inicjacji tego trybu należy zainstalować inny sterownik,dający taką możliwość. Na rynku można spotkać takie sterowniki,jak *VGA256.BGI* czy *SVGA256.BGI*.
Oto procedura instalująca dowolny sterownik i uruchamiająca dowolny tryb graficzny :

```
Procedure GrafikaV(sterownik:string;tryb:integer);
var autoDetect :pointer;
```

```
    s1,s2:word;
{$F+}
  Function Nowy:integer;
   begin
     Nowy:=GrMode;
  end;
{$F-}
BEGIN
  GrMode:=tryb;
  autoDetect:=@Nowy;
  GrDriver:=InstallUserDriver(sterownik,autoDetect);
  if GraphResult<>grOK then
     begin
       writeln('Blad ladowania');
       Halt(1);
     end;
     GrDriver:=Detect;
  InitGraph(GrDriver,GrMode,'');
  GetAspectRatio(s1,s2);
   skala:=s1/s2;
END;
```

Podobnie jak w poprzedniej procedurze zmienne *GrDriver* i *GrMode* są zmiennymi globalnymi.
Dla sterownika *VGA256.BGR* tryb 320x200 i 256 kolorów uzyskamy wywołując procedurę :
GrafikaV('vga256',0); (tryb nr 0 jest zresztą jedynym ,jaki daje ten sterownik).
Natomiast dla sterownika *SVGA256.BGI* wywołanie powinno być następujące :
GrafikaV('svga256',0);
Sterownik *SVGA256.BGI* pozwala zainicjować grafikę również w innych trybach, o ile dysponujemy kartą SVGA.Eksperymenty pozostawiamy Czytelnikowi,zwłaszcza że możliwy do uzyskania tryb zależy od indywidualnych cech posiadanej karty graficznej.
Po prawidłowym zainstalowaniu sterownika i inicjacji trybu graficznego procedury i funkcje modułu *GRAPH* są do naszej dyspozycji. Istnieje wiele różnych opini na temat tego modułu.

II.2.Wykorzystanie BIOSU i adresowania do pamięci.

2.a. Uruchomienie grafiki

Nie wdając się w ocenę tych opinii,zajmiemy się teraz grafiką trybu 320x200,256 kolorów bez pośrednictwa modułu *GRAPH.*Pokażemy proste zastosowania obsługi tego trybu (**13h**) przez BIOS i poprzez bezpośredni dostęp do pamięci ekranu.
Posłużymy się modelem,nie wymagającym od Czytelnika zaawansowanych umiejętności programistycznych,niezbędna jest jednak znajomość w podstawowym zakresie języka Pascal oraz architektury komputerów PC.
Należy zaznaczyć jeszcze,że niniejsze rozwiązanie będzie funkcjonowało poprawnie na komputerach wyposażonych w kartę VGA lub dowolną kartę SVGA. Zacznijmy więc od przeglądu możliwości .VGA BIOS definiuje następujące tryby graficzne:

Numer	Rozdzielczość	Liczba kolorów
04h	320x200	4
05h	320x200	4
06h	640x200	2
0Dh	320x200	16
0Eh	640x200	16
0Fh	640x350	4
10h	640x350	16
11h	640x480	2
12h	640x480	16
13h	320x200	256

Oto procedura, inicjująca interesujący nas tryb graficzny 13h:

```
Program Rysuj;
uses dos;
Procedure Grafika;
var rej :Registers;
begin
  rej.AX:=$0013;
  INTR($10,rej);
end;
 ....
```

2.b. Rysowanie punktu i testowanie koloru punktu

Mamy, co prawda, tryb graficzny,ale by narysować cokolwiek na ekranie,potrzebne jest nam pierwsze narzędzie graficzne.
W trybach graficznych najmniejszym, a zarazem podstawowym obiektem jest punkt na ekranie,tzw piksel.Jak się dalej okaże, będzie to nasze podstawowe i właściwie jedyne narzędzie,przy pomocy którego skonstruujemy następne.
Podobnie jak w trybach tekstowych,tak i w graficznych BIOS dzieli dostępną pamięć ekranu na strony.Jedna ze stron jest zawsze widoczna,pozostałe niewidoczne,ale można na nich wykonywać dowolne operacje.Liczba stron zależy od trybu i rozmiaru pamięci RAM zainstalowanej na karcie.W trybie 13h na zapamiętanie pełnego ekranu potrzeba 64kB pamięci i niestety, mamy dostęp tylko do jednej strony.
BIOS zapewnia realizację rysowania punktu w żądanym kolorze oraz operację odwrotną , tzn odczytu koloru wskazanego punktu.Oto procedura rysująca punkt o danych współrzędnych całkowitych :

```
Procedure Kropka(x,y:integer;kolor:byte);
var rej :registers;
begin
    With Rej do
       begin
         AH:=$0C;
         AL:=kolor;
         CX:=x;
         DX:=y;
       end;
       INTR($10,Rej);
end;
```

oraz funkcja testująca kolor punktu :

```
Function Kolor_Kropki(x,y:integer):byte;
var rej :registers;
begin
    With rej do
        begin
            AH:=$0C;
            CX:=x;
            DX:=y;
            BH:=0;          {numer strony,w trybie 13h zawsze 0}
        end;
      INTR($10,rej);
      Kolor_Kropki:=rej.AL;
end;
```

W powszechnej opinii szybkość realizacji operacji graficznych przez BIOS pozostawia jednak wiele do życzenia i tam , gdzie to będzie możliwe, zastosujemy inne rozwiązanie. Jak już wspominaliśmy , na zapamiętanie jednego ekranu potrzeba 64000 bajtów.
Ekran ma rozdzielczość 320x200 punktów,każdy punkt jest opisany przez 1 bajt (8 bitów, co daje możliwość opisania każdego z 256 kolorów).Każdej z 200 linii odpowiada więc 320 bajtów,zaś pamięć ekranu rozpoczyna się od adresu $A000:0,co oznacza , że zawartość
linii numer 0 znajduje się od adresu $A000:0 do adresu $A000:$013F,
linii numer 1 od adresu $A000:$0140 do adresu $A000:$027F itd.
Stąd zapis o kolorze punktu P(x,y) znajdziemy pod adresem **$A000:320*y+x.**
Punkt na ekranie można uzyskać , wpisując bezpośrednio do odpowiedniego adresu pamięci wartość koloru tego punktu i tak:

```
Procedure Punkt(x,y:integer;kolor:byte);
begin
    Mem[$A000:320*y+x]:=kolor;
end;
```

zaś odczytać kolor punktu,odczytując zawartość odpowiedniej komórki pamięci:

```
Function Kolor_Punktu(x,y:integer):byte;
begin
    Kolor_Punktu:=Mem[$A000:320*y+x];
end;
```

Przyjmiemy ten drugi sposób z pewną zmianą.Aby prześpieszyć realizację rysowania punktu, wyeliminujemy obliczenia adresu względnego. Posłużymy się zmienną adresowaną do pamięci,a dokładniej tablicą adresowaną do pamięci.

```
type Ekran=array[0..199,0..319] of byte;
var scr : Ekran absolute $A000:0;
```

Powyższe procedury przyjmą teraz następującą postać :

```
Procedure Piksel(x,y:integer;kolor:byte);
begin
    scr[y,x]:=kolor;
end;
Function Kolor_Piksla(x,y:integer):byte;
begin
    Kolor_Piksla:=scr[y,x];
end;
```

Ponadto zastosowana zmienna *scr* okaże się niezwykle użyteczna przy realizacji innych zadań graficznych.Wydawać się może,że taki sposób postępowania jest nieracjonalny.Zmienna *scr*,ze względu na swój rozmiar powinna blokować praktycznie prawie cały obszar przeznaczony dla zmiennych w programie.Otóż zaadresowanie jej do pamięci eliminuje te koszty,w dalszym ciągu mamy 64 kB dla innych zmiennych !
Spróbujmy wykonać prosty rysunek przy pomocy sformułowanych już procedur:

```
Program Rysunek1;
uses crt,dos;
type Ekran=array[0..199,0..319] of byte;
var scr : Ekran absolute $A000:0;
       i:integer;
Procedure Piksel(x,y:integer;kolor:byte);
begin
    scr[y,x]:=kolor;
end;

Procedure Grafika;
var rej :Registers;
begin
  rej.AX:=$0013;
  INTR($10,rej);
end;
BEGIN
    Grafika;
    for i:=0 to 400 do Piksel(i,i,i mod 256);
    Repeat Until KeyPressed;
    TextMode(3);    { Powrót do trybu tekstowego,koniec trybu graficznego }
END.
```

Na ekranie uzyskamy efekt , od którego odzwyczaiły nas procedury modułu *GRAPH,* mianowicie punkty,których współrzędne wykraczają poza zakres ekranu (320x200,środek układu w lewym górnym rogu), są jednak rysowane na zasadzie "sklejonego" lub "zawiniętego" ekranu.Zachodzi zatem konieczność uzbrojenia procedury **Piksel** w mechanizm obcinania do brzegów ekranu (tzw clipping).Ponadto z problemem tym spotkamy się przy okazji rysowania innych obiektów jak odcinek czy okrąg.Ponieważ inne figury będziemy konstruować właśnie z punktów, sprawa jest istotna. Najprostsze wydaje się być następujące rozwiązanie :

```
Procedure Punkt(x,y:integer;kolor:byte);
begin
  if ((x>-1) and (x<320) and (y>-1) and (y<200)) then
    scr[y,x]:=kolor;
end;
```

Po wprowadzeniu tej poprawki Program Rysuj1 daje linię prostą o początku w punkcie (0,0) i końcu (199,199),bez niepożądanych efektów.Procedurę rysującą dowolny odcinek omówimy szczegółowo za chwilę.Po uruchomieniu programu i zainicjowaniu trybu graficznego ekran pozostaje w kolorze czarnym (punkty możemy rysować różnymi kolorami z zakresu 0÷255).
Jak uzyskać efekt zmiany tła całego ekranu i to z zadawalającą szybkością ?
Pożądanej szybkości może nie zapewnić takie np rozwiązanie:

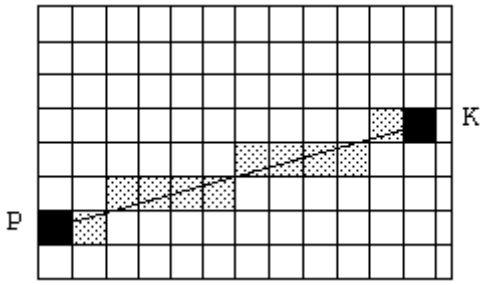
```
for i:=0 to 199 do
    for j:=0 to 319 do scr[i,j]:=0;
```

Wykorzystamy tutaj zalety zmiennej *scr* oraz posłużymy się procedurą
FillChar(*cel,ile,czym*),
która wypełnia obszar pamięci zlokalizowany przez zmienną *cel*,*ile* zmiennymi bajtowymi o wartościach *czym,*która powinna zapewnić szybką realizację tego zadania :

```
Procedure Tlo(kolor:word);
 begin
   FillChar(scr,64000,kolor);
 end;
```

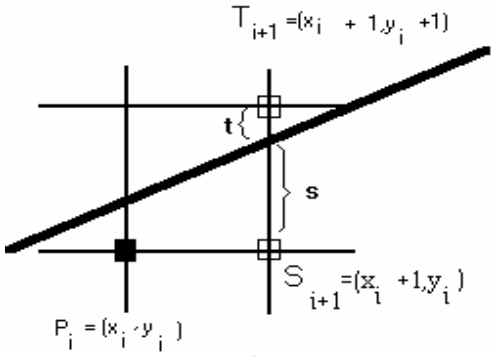
2.c. Rysowanie odcinka

Następny problem,który rozwiążemy,to rysowanie odcinka z punktu o współrzędnych(x_p , y_p)do punktu o współrzędnych (x_k, y_k).BIOS, niestety ,nie udostępnia żadnych mechanizmów rysujących bardziej złożone obiekty, np proste czy okręgi.Przy konstrukcji procedury rysującej odcinek możemy jedynie skorzystać z już zdefiniowanego **Punktu.**Podobnie postąpimy rozwiązując inne zadania graficzne.Najbardziej popularnym i najczęściej stosowanym algorytmem rysowania odcinków jest algorytm zaproponowany w 1963 roku przez Bresenhama.



rys.1

Rysując dcinek ***PK*** musimy przejść do układu współrzędnych całkowitych,tak by przy pomocy ograniczonej liczby piksli przybliżyć ciągły odcinek prostej.Zmusza to nas do wyboru rysowanych punktów ekranu na drodze z ***P*** do ***K.***Rysunek zaczniemy oczywiście od narysowania punktu $P(x_p, y_p)$.Załóżmy na razie,że początek $P(x_p, y_p)$ odcinka i jego koniec $K(x_k, y_k)$ całkowicie mieszczą się na ekranie oraz $x_p \langle x_k$



rys.2

Musimy zdecydować teraz,który z punktów ***T*** czy ***S*** będzie następnym w naszym odcinku.Tego typu wyboru należy dokonywać do momentu osiągnięcia punktu końcowego $K(x_k, y_k)$.Jako kryterium wyboru kolejnych punktów ekranu przyjmiemy długości odcinków *t* i *s*.
Jeśli *s>t*,następnym punktem odcinka będzie ***T***,w przeciwnym przypadku rysujemy ***S***.
Na ogół długości *s* oraz *t* będą liczbami rzeczywistymi.Spróbujemy ustalić związek między *s* i *t*,tak by algorytm nie był uwikłany w zawiłe rachunki zmiennopozycyjne.

Oznaczmy ***dx=xk-xp*** oraz ***dy=yk-yp***.

Kolejne punkty odcinka powinny spełniać równanie (1) $y - y_p = \frac{dy}{dx}(x - x_p)$.

Wstawiamy współrzędne kolejnego punktu odcinka $(x_{i+1}, y_i + s)$ do równania (1), otrzymując kolejno :

$$y_i + s - y_p = \frac{dy}{dx}(x_{i+1} - x_{p)}$$

oraz

$$s = \frac{dy}{dx}(x_{i+1} - x_p) - (y_i - y_p)$$

(2) $dx \cdot s = dy(x_{i+1} - x_p) - (y_i - y_p) \cdot dx$ a po wymnożeniu obu stron przez *dx*

Zauważmy,że ***t=1-s*** ,co pomnożone obustronnie przez *dx* daje ***dx·t=dx-dx·s.*** (3)

Zbadamy wyrażenie

dx·s-dx·t=*dx(s-t)=dx·s-dx+dx·s*=***2dx·s-dx*** na podstawie (3)

oznaczymy następnie ***dx·(s-t)***=d_i

Z równania (2) otrzymujemy

$$d_i = 2dy(x_{i+1} - x_p) - 2dx(y_i - y_p) - dx$$

oraz

$$d_{i+1} = 2dy(x_{i+2} - x_p) - 2dx(y_{i+1} - y_p) - dx$$

czyli

$$d_{i+1} - d_i = 2dy(x_{i+2} - x_{i+1}) - 2dx(y_{i+1} - y_i)$$

ponieważ

$x_{i+2} - x_{i+1} = 1$ dla każdego kroku postępowania,otrzymujemy wyrażenie,które posłuży jako kryterium wyboru punktu ***T*** lub ***S*** w procedurze rysującej odcinek :

$$\boxed{d_{i+1} = d_i + 2dy - 2dx(y_{i+1} - y_i)}$$

ponadto zauważmy,że jeśli

1° $d_i \geq 0$ tzn $s>t$ i $y_{i+1} - y_i = 1$ czyli rysujemy punkt punkt ***T***

zaś

(4) $\boxed{d_{i+1} = d_i + 2(dy - dx)}$

2° $d_i < 0$ tzn $s<t$ i $y_{i+1} - y_i = 0$ czyli rysujemy punkt ***S***

zaś

(5) $\boxed{d_{i+1} = d_i + 2dy}$

Związki (4) i (5) będą roztrzygać o wyborze kolejnych piksli na drodze ***PK***.Ponieważ są to wzory rekurencyjne,jako początkową wartość przyjmiemy d_0=***2dy-dx.***

Oto implementacja powyższego algorytmu :

```
Procedure Kreska(xp,yp,xk,yk,kolor :integer);
var dx,dy,d,x,y,d1,d2,sx,sy :integer;
begin
  dx:=abs(xk-xp);  dy:=abs(yk-yp);
  if xk>=xp then sx:=1 else sx:=-1;
  if yk>=yp then sy:=1 else sy:=-1;
  x:=xp;
  y:=yp;
  if dx>=dy then
     begin
       d:=2*dy-dx;
       d1:=2*dy;
       d2:=2*(dy-dx);
       scr[y,x]:=kolor;
       While (x<>xk) do
          begin
            if d<0 then
              begin
                Inc(d,d1);
                Inc(x,sx);
              end
                else
                   begin
                     Inc(x,sx);
                     Inc(y,sy);
                     Inc(d,d2);
                   end;
             scr[y,x]:=kolor;
           end
      end
        else
```

```
          begin  {dx<dy}
            d:=2*dx-dy;
            d1:=2*dx;
            d2:=2*(dx-dy);
            scr[y,x]:=kolor;
            while (y<>yk) do
              begin
                if d<0 then
                  begin
                    Inc(d,d1);
                    Inc(y,sy);
                  end
                    else
                      begin
                        Inc(x,sx);
                        Inc(y,sy);
                        Inc(d,d2);
                      end;
                scr[y,x]:=kolor;
              end
          end;
end;
```

W procedurze rozpatrzono dwa przypadki 1° *dx>dy* i 2° *dx<dy* .Takie podejście pozwala rysować "z kątem nachylenia" odcinka z przedziału $\langle 0, \frac{\Pi}{4} \rangle$,co gwarantuje dobrą jakość w każdym przypadku.Niestety, problemu nie rozwiązaliśmy jeszcze całkowicie.W procedurze **Kreska**,polecania *scr[y,x]:=kolor* konstruują całą linię ***PK***,co jak zauważyliśmy wyżej,działa poprawnie jedynie w sytuacji,gdy punkty ***P*** i ***K*** leżą w obrębie ekranu.
Odpowiednio jednak postępując tzn,dbając o to by współrzędne początku i końca odcinka mieściły się na ekranie,możemy już skorzystać z procedury **KRESKA**.W następnym rozdziale zajmiemy sie szczegółowo problemem okienkowania (tzw clippingiem) odcinka.
Oto prosty program sprawdzający poprawność dotychczasowego rozwiązania na przykładzie animacji metodą zamazywania figury kolorem tła.Załóżmy,że w module **GRAF_13** mamy zgromadzone procedury : *GRAFIKA, KRESKA* oraz *SYNCHRONIZACJA*, która jest zdefiniowana następująco:

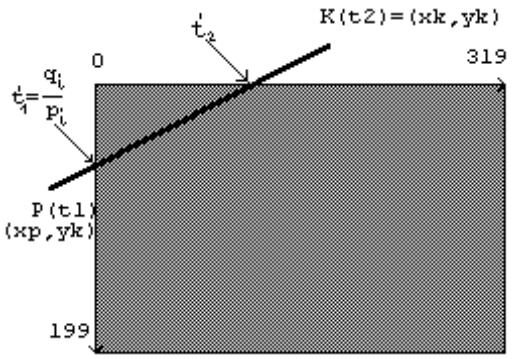
```
Procedure Synchronizacja;
 begin
   While (Port[$3DA] and 8)=0 do
 end;
```

Na ekranie chcemy uzyskać efekt bilardu na prostokątnym stole, gdzie rolę odbijanej kuli pełni kwadrat. Zdefiniowane funkcje *OK_X* oraz *OK_Y* gwarantują, że procedura *KRESKA* otrzyma współrzędne zawierające się w ekranie oraz zapewnią efekt "uginania się" figury przy zetknięciu z brzegami ekranu.Procedura *SYNCHRONIZACJA* poprawi elegancję powstawania rysunku w nowej fazie.

```
Program Animacja_kwadratu; { metodą zmiany koloru }
uses crt,graf_13;
const a=25;
      b=20;
var x,y :integer;
Function Ok_X(p:integer):integer;
begin
  if p<1 then Ok_X:=1
    else
      if p>318 then Ok_X:=318
```

```
        else Ok_X:=p;
end;
Function Ok_Y(q:integer):integer;
begin
  if q<40 then Ok_Y:=40
    else
      if q>198 then Ok_Y:=198
        else Ok_Y:=q;
end;
Procedure Prostokat(kolor:byte);
var i:byte;
begin
  for i:=0 to 1 do
    begin
      Kreska(Ok_X(x+a*(2*i-1)),Ok_Y(y),Ok_X(x),Ok_Y(y+b*(2*i-1)),kolor);
      Kreska(Ok_X(x),Ok_Y(y+b*(2*i-1)),Ok_X(x-a*(2*i-1)),Ok_Y(y),kolor);
    end;
end;
Procedure Bilard;
var p_x,p_y:integer;
begin
  x:=160;y:=100;
  for p_x:=0 to 1 do        { rysuj ramkę }
    begin
      Kreska(p_x*319,39,p_x*319,199,15);
      Kreska(0,39+p_x*160,319,39+p_x*160,15);
    end;
  p_x:=2; p_y:=2;
  Repeat
    Prostokat(14);                              {narysuj nowy rysunek}
    if ((x>340) or (x<-20)) then p_x:=-p_x;     { odbij od brzegów poziomych}
    if ((y>215) or (y<20)) then p_y:=-p_y;      { odbij od brzegów pionowych }
    Synchronizacja;
    Prostokat(9);                               {zmaż poprzedni rysunek}
    Inc(x,p_x);                                 {przygotuj nową fazę}
    Inc(y,p_y);
  Until KeyPressed;
end;

BEGIN
  Tlo(9);
  Bilard;
  TextMode(3);
END.
```

2.d. Obcinanie odcinka do brzegów ekranu



rys.3

Gdyby w procedurze **Kreska** polecenie *scr[y,x]:=kolor* zastąpić wywołaniem *Punkt(x,y,kolor),* efekt okienkowania zostanie oczywiście osiągnięty. Koszt takiego rozwiązania może być jednak znaczny,zwłaszcza w sytuacji,kiedy będziemy dynamiczne rysowali większą liczbę odcinków, np podczas animacji.**Punkt** sprawdza, czy spełniony jest układ nierówności : `if ((x>-1) and (x<320) and (y>-1) and (y<200)) then ...`
nawet wówczas, gdy nie ma takiej potrzeby,rysując

więc np odcinek *P=(0,0),K=(319,199)* dokonywana jest zbędna operacja aż 199 razy,oprócz,rzecz jasna niezbędnych obliczeń w samym algorytmie Bresenhama. Zastosujemy rozwiązanie zaproponowane w znanym algorytmie **Lianga -Barsky'ego.** Chcąc zapewnić algorytmowi dynamikę w sytuacji,jak na rys 3,wyznaczymy ewentualne nowe końce odcinka,tak by rysować tylko tę część odcinka,która widoczna jest na ekranie. Odcinek zadany jest przez współrzędne początku i końca,wobec tego możliwa jest reprezentacja za pomocą równania parametrycznego.

oznaczmy ***px=xk-xp***

i ***py=yk-yp.***

Otrzymujemy następujące równanie parametryczne odcinka ***PK*** :

$$\begin{cases} x = x_p + t \cdot p_x \\ y = y_p + t \cdot p_y \end{cases}$$

gdzie punkt ***P*** otrzymujemy dla parametru *t*=0,zaś koniec ***K*** dla parametru *t*=1.Pozostałe punkty odcinka ***PK*** dla $t \in (0,1)$.

Algorytm Lianga-Barsky'ego wyznacza nie punkty przęcięcia z brzegami ekranu,lecz odpowiadające tym punktom wartości parametru *t*,na rys.3 t_1' i t_2'.Dowolny punkt (*x,y*) odcinka *PK* zawiera się w ekranie,jeśli dla parametru *t* wyznaczającego punkt (*x,y*) spełniony jest układ nierówności :

$$\begin{cases} 0 \le x_p + t \cdot p_x \le 319 \\ 0 \le y_p + t \cdot p_y \le 199 \end{cases}$$

lub inaczej:

$$(i)\begin{cases} -t \cdot p_x \le x_p & \text{dla lewego brzegu ekranu} \\ t \cdot p_x \le 319 - x_p & \text{dla prawego brzegu} \\ -t \cdot p_y \le y_p & \text{dla górnego brzegu naszego ekranu itd.} \\ t \cdot p_y \le 199 - y_p & \end{cases}$$

Układ nierówności (i) zapiszemy ogólnie jako :

$$(ii)\ \boxed{t \cdot p_i \le q_i}\ .$$

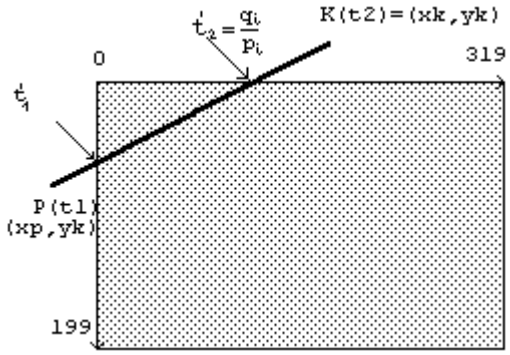
Nasuwa się oczywisty wniosek,że dla $t = \frac{q_i}{p_i}$ otrzymamy punkt przecięcia odcinka ***PK*** z krawędzią ekranu.Aby więc wyznaczyć nowe końce odcinka, należy wykonać test dla czterech brzegów ekranu.W każdym teście ponadto rozpatrzymy sytuacje :

1° jeśli p_i=0,oznacza to, że odcinek *PK* jest równoległy do testowanego brzegu (dla i=1,2 odpowiednio dla lewego i prawego,dla i=3,4 dla górnego i dolnego).

2° jeśli p_i<0 należy szukać punktu przecięcia z brzegiem od strony punktu *P*, w przeciwnym przypadku od strony punktu *K*.

Jeśli ponadto q_i<0,to odcinek leży całkowicie poza ekranem,w przeciwnym przypadku należy wyznaczać punkty przecięcia.

Test zaczniemy dla t_1=0 i t_2=1,po teście dla jednego z brzegów ekranu zweryfikowane wartości t_1 i t_2,posłużą jako wartości wyjściowe dla testów z natępnymi brzegami.



rys.4

Rysunek 3 przedstawia test lewego brzegu,dla którego $p_i = -(x_k - x_p) \le 0$, dlatego punkt przecięcia znaleziono od strony punktu ***P***.
Na rysunku 4 testując górny brzeg mamy odwrotną sytuację ponieważ $p_i = -(y_k - y_p) \ge 0$ i nowego końca szukamy od strony punktu ***K***.
Oto odpowiednia funkcja testująca :

```
{------------Test-Lianga_Barsky'ego---------------}
Function Brzeg(p,q :integer;var t1,t2:real):boolean;
var fakt:boolean;
       t:real;
begin
  fakt:=true;
  if p<0 then
    begin
       t:=q/p;
       if t>t2 then fakt:=false
         else
          if t>t1 then t1:=t;        { nowy koniec od strony P }
    end
    else
  if p>0 then
    begin
       t:=q/p;
       if t<t1 then fakt:=false
         else
          if t<t2 then t2:=t;       { nowy koniec od strony K }
    end
     else
  if q<0 then fakt:=false;          { odcinek leży poza ekranem }
  Brzeg:=fakt;
end;
```

Ostatecznie procedura rysująca odcinek przyjmie następującą postać :

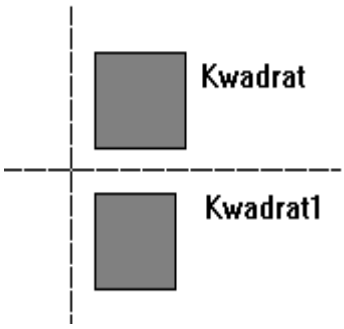
```
Procedure Linia(xp,yp,xk,yk,kolor:integer);
var t1,t2:real;
      px,py,x,y:integer;
begin
  px:=xk-xp;
  py:=yk-yp;
  x:=xp;y:=yp;
  t1:=0;t2:=1;
  if Brzeg(-px,xp,t1,t2) then           { wykonaj test dla lewego brzegu }
    if Brzeg(px,319-xp,t1,t2) then      { wykonaj test dla prawego brzegu }
      if Brzeg(-py,yp,t1,t2) then       { wykonaj test górnego brzegu }
        if Brzeg(py,199-yp,t1,t2) then  { wykonaj test dla dolnego brzegu }
        begin                           { oblicz nowe końce odcinka }
          if t1>0 then
            begin
              xp:=Round(xp+t1*px);
              yp:=Round(yp+t1*py);
            end;
          if t2<1 then
            begin
```

```
            xk:=Round(x+t2*px);
            yk:=Round(y+t2*py);
          end;
        Kreska(xp,yp,xk,yk,kolor);   { rysuj odcinek }
      end;
end;
```

2.e. Rysowanie okręgu i elipsy

W tym rozdziale zajmiemy się procedurą rysującą okrąg. Zaczniemy jednak od przykładu, który ilustruje dodatkowy problem do przezwyciężenia,oprócz samego rysowania okręgu.Jak się okaże dalej,pozwoli to nam tak skonstruować procedurę rysującą okrąg,by przy jej pomocy rysować również dowolną elipsę.
Zacznijmy jednak od narysowania kwadratu o bokach równoległych do krawędzi ekranu:

```
Procedure Kwadrat1(x,y,bok:integer;kolor:byte);
begin
  Linia(x,y,x+bok,y,kolor);
  Linia(x+bok,y,x+bok,y+bok,kolor);
  Linia(x+bok,y+bok,x,y+bok,kolor);
  Linia(x,y+bok,x,y,kolor);
end;
```



rys.6

Rysunek 6 ilustruje skutek wywołania procedury:
Kwadrat1*(40,80,40,0)*.
Widać niestety różnicę w długości boków poziomych i pionowych. Towarzyszący układ współrzędnych jest tak pomyślany, że odcinki na osiach są równe "długości" 10 piksli osi OY.Mimo poprawnej konstrukcji procedury *Kwadrat1* na ekranie wyraźnie widoczny jest prostokąt.Powodem tego zniekształcenia jest to, że piksle ekranu nie są kwadratowe. Zjawisko to charakteryzuje tzw. aspekt,który będzie nam towarzyszył również przy konstruowaniu procedury rysującej okrąg i elipsę. Aspekt definiuje się jako ***stosunek odległości środków sąsiednich piksli w poziomie do odległości sąsiednich piksli w pionie***.
Dla trybu 13h wielkość aspektu określa ułamek $a = \frac{22}{25}$ .Aby uzyskać więc kwadrat pozbawiony zniekształcenia, długość boków poziomych należy podzielić przez wielkośc *a* aspektu.
Oto zmodyfikowana procedura rysująca kwadrat:

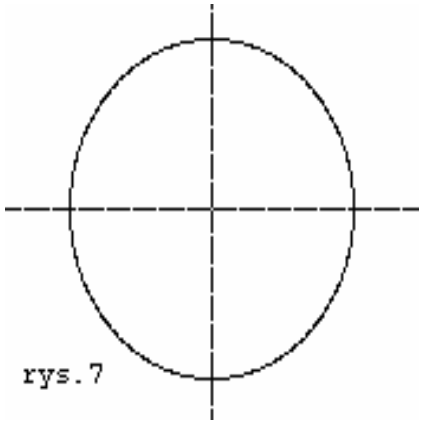
```
Procedure Kwadrat(x,y,bok:integer;kolor:byte);
begin
  Linia(x,y,x+Round(q/p*bok),y,kolor);
  Linia(x+Round(q/p*bok),y,x+Round(q/p*bok),y+bok,kolor);
  Linia(x+Round(q/p*bok),y+bok,x,y+bok,kolor);
  Linia(x,y+bok,x,y,kolor);
end;
```

gdzie *p*=22 zaś *q*=25.Na rysunku 6 widzimy skutek poprawionej procedury wywołanej:
Kwadrat*(40,20,40,0)*.
Z podobnym zjawiskiem spotkamy się oczywiście przy rysowaniu okręgu.Spróbujmy najpierw narysować okrąg wykorzystując jego równanie parametryczne :
[a+r·sin(k),b+r·cos(k)], gdzie *(a,b)* to środek okręgu,*r* - promień.

```
Procedure Okrag_(a,b,r:integer;kol:byte);
var k,dk:real;
```

```
    x,y:integer;
  Procedure RysPkt(xo,yo,x,y:integer;k:word);
  begin
      Punkt(xo+x,yo+y,k);Punkt(xo-x,yo+y,k);
      Punkt(xo+x,yo-y,k);Punkt(xo-x,yo-y,k);
  end;

begin
  k:=Pi/2; dk:=0.01;          { na ogół przyjmuje się dk=1/r}
  Repeat
    x:=Round(r*cos(k));
    y:=Round(r*sin(k));
    RysPkt(a,b,x,y,kol);
    k:=k-dk;
  Until k<0;
end;
```

Dowolny algorytm nie uwzględniający aspektu,da w wyniku na ekranie rysunek w postaci elipsy (patrz rysunek 7).

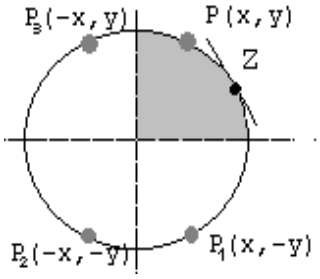


rys.7

Kolejna wada tego rozwiązania to niska jakość rysunku, spowodowana zaokrągleniami do liczb całkowitych. Konstrukcja jaką teraz zaproponujemy,powinna więc posługiwać się tylko obliczeniami na liczbach całkowitych oraz uwzględniać aspekt ekranu.Zakładając zaś,że potrafimy narysować okrąg (nie wyglądający jak elipsa),możemy odpowiednio zmieniając aspekt,z okręgu uzyskać elipsę.
Rysując okrąg będziemy więc posługiwali się dwoma układami współrzędnych - układem pikslowym (0xy) i układem rzeczywistym(0XY).
Przy czym $x=X/a$ oraz $y=Y$ gdzie a jest wielkością aspektu (w naszej sytuacji $a=\frac{22}{25}$).W układzie rzeczywistym (z uwzględnionym aspektem) okrąg pozbawiony jest wad,o których wspominaliśmy,niemniej jednak rysować musimy w układzie pikslowym.Załóżmy na razie,że środek okręgu znajduje się w punkcie (0,0),zaś promień r jest liczbą naturalną.Okrąg to krzywa o równaniu :

$F(x,y)$: $X^2+Y^2-r^2=0$ (w układzie rzeczywistym)

lub $\left(\frac{p}{q}x\right)^2+y^2-r^2=0$ (w układzie pikslowym).



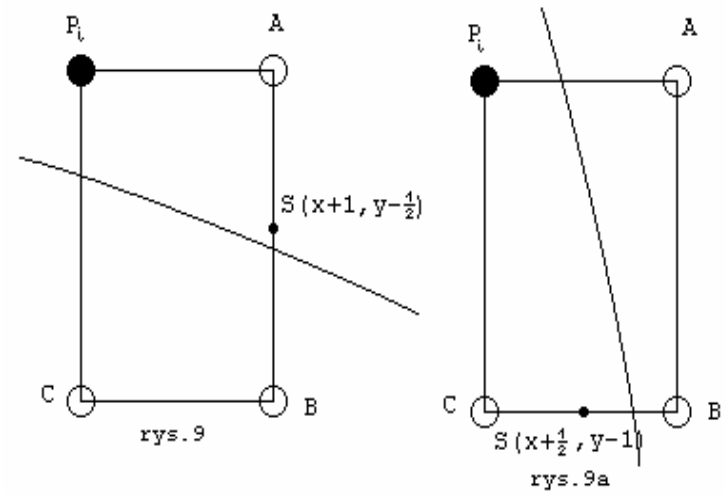
rys.8

Rysunek zaczniemy od punktu (0,r) i wystarczy wyznaczyć punkty okręgu pierwszej ćwiartki,na drodze do punktu (r,0).Pozostałą część narysujemy wykorzystując symetrię okręgu (patrz rysunek 8).
Podobnie jak w algorytmie rysującym odcinek i tu będziemy zmuszeni dokonywać wyboru kolejnych piksli do narysowania.
W tym celu wyznaczymy funkcję oceniającą,której wartości będą naliczane rekurencyjnie w trakcie działania procedury.Ponadto sposób wybierania kolejnych piksli okręgu będzie musiał być różny w dwóch częściach drogi jaką mamy do przebycia.Wynika to zarówno z charakteru okręgu jak i wspomnianego już aspektu ekranu.Rysunek 9 ilustruje zasadę wybierania kolejnego punktu w części,nazwijmy ją chwilowo,bliższą punktowi z którego zaczęliśmy rysować.Załóżmy,że narysowaliśmy już punkt $P_i(x,y)$.Następnym będzie *A(x+1,y)* lub *B(x+1,y-1)*.Na rysunku zaś 9a pokazana jest zasada wyboru kolejnego punktu okręgu w części "bliższej" punktowi *(r,0)*.Wyboru dokonamy spośród *B(x+1,y-1)* i *C(x,y-1)*.

Jako granicę obierzemy taki punkt Z (patrz rys.8) okręgu,dla którego współczynnik wektora stycznego do okręgu wynosi -1.
Obliczymy w tym celu pochodną zmodyfikowanej funkcji *F(x,y)* do postaci :

$$F(x,y) = p^2x^2 + q^2y^2 - q^2r^2.$$

(1)



Powodem tej modyfikacji jest konieczność posługiwania się obliczeniami całkowitoliczbowymi. Pochodną takiej funkcji obliczymy posługując się następującym twierdzeniem:

$$\boxed{\tfrac{d}{dx}F(x,y) = F_x \cdot 1 + F_y \cdot y'}$$

i dalej w naszej sytuacji:
F(x,y)=0 gdzie *y=y(x)*;
czyli
F(x,y(x))=0 i po zróżniczkowaniu obu stron

$$F_x(x, y(x)) + F_y(x, y(x)) \cdot y' = 0$$

z ostatniej równości wystarczy wyznaczyć y' :

$$y' = -\tfrac{F_x}{F_y}$$

co dla funkcji postaci (1) daje wyrażenie : $\boxed{y' = -\frac{2p^2x}{2q^2y}}$.Oczywiście wyliczanie dokładnych współrzędnych punktu Z nie jest konieczne.Pierwszy sposób wyboru kolejnych punktów okręgu (rys.9) zastosujemy wtedy gdy stwierdzimy prawdziwość nierówności $p^2x < q^2y$ w przeciwnym przypadku sposób pokazany na rysunku 9a.Przejdźmy teraz do ustalenia funkcji oceniającej.Będzie to wartość funkcji *F* w punkcie środkowym *S* (patrz rys.9 i 9a) między alternatywnymi pikslami.I tak dla pierwszego fragmentu okręgu (rys.9) obliczymy

$$F_{s_i} = F(x_i + 1, y_i - \tfrac{1}{2}) = p^2(x_i + 1)^2 + q^2(y_i - \tfrac{1}{2})^2 - q^2r^2 \quad \text{(I)}$$

przy czym jeśli $F_{s_i} \geq 0$ wybierzemy punkt *B* w przeciwnym przypadku punkt *A*.
Dla drugiego fragmentu obliczymy
$F_{s_i} = F(x_i + \tfrac{1}{2}, y_i - 1)$
i dla $F_{s_i} \geq 0$ rysujemy punkt *B* wpp rysujemy punkt *C*.
Dla punktu startowego $P_0 = (0, r)$ wartość funkcji oceniającej wynosi :
$f_{s_0} = (x_0 + 1, y_0 - \tfrac{1}{2}) = p^2(0+1)^2 + q^2(r - \tfrac{1}{2})^2 - qR^2$ zaś po wykonaniu obliczeń :
$f_{s_0} = p^2 - q^2r + \tfrac{1}{4}q^2$ .Ponieważ w algorytmie chcemy mieć obliczenia całkowitoliczbowe,ostatnią równość pomnóżmy stronami przez 4.Otrzymamy wzór :

$$\boxed{4f_{s_0} = 4p^2 - 4q^2r + q^2} \quad \text{(II)}$$

Wyprowadźmy teraz wzory dla funkcji oceniającej w pierwszej części okręgu w przypadku wyboru jako następnego punktu $P_{i+1} = A = (x_{i+1}, y_{i+1}) = (x_i + 1, y_i)$.
W tym celu obliczymy :

$$\begin{aligned} F(x_{i+1} + 1, y_{i+1} - \tfrac{1}{2}) &= p^2(x_{i+1} + 1)^2 + q^2(y_{i+1} - \tfrac{1}{2})^2 - q^2r^2 = \\ &= p^2(x_i + 1 + 1)^2 + q^2(y_i - \tfrac{1}{2})^2 - q^2r^2 = \\ &= p^2(x_i + 1)^2 + 2p^2(x_i + 1) + p^2 + q^2(y_i - \tfrac{1}{2})^2 - q^2r^2 \end{aligned}$$

a po uwzględnieniu (I) i współrzędnych punktu *A*,równość przyjmie postać:
$f(x_{i+1} + 1, y_{i+1} - \tfrac{1}{2}) = f(x_i + 1, y_i - \tfrac{1}{2}) + 2p^2x_{i+1} + p^2$,którą krótko zapiszemy:

$$f_{s_{i+1}} = f_{s_i} + 2p^2 x_{i+1} + p^2$$

lub $\boxed{4f_{s_{i+1}} = 4f_{s_i} + 8p^2 x_{i+1} + 4p^2}$ (III)

Jeśli w pierwszej części będziemy zmuszeni jako kolejny,wybrać punkt $P_{i+1} = B = (x_{i+1}, y_{i+1}) = (x_i + 1, y_i - 1)$,obliczymy:

$$F(x_{i+1}+1, y_{i+1} - \tfrac{1}{2}) = p^2(x_i + 1 + 1)^2 + q^2(y_i - \tfrac{1}{2} - 1)^2 - q^2 r^2$$
$$= p^2(x_i+1)^2 + 2p^2(x_i+1) + p^2 + q^2[(y_i - \tfrac{1}{2}) - 2(y_i - \tfrac{1}{2}) + 1] - q^2 r^2 =$$
$$= p^2(x_i+1)^2 + 2p^2(x_i+1) + p^2 + q^2[(y_i - \tfrac{1}{2}) - 2(y_i - 1)] - q^2 r^2,$$

uwzględniając zaś (I) i współrzędne punktu *B* otrzymujemy :

$$f_{s_{i+1}} = f_{s_i} + 2p^2 x_{i+1} + p^2 - 2q^2 y_{i+1}$$

inaczej $\boxed{4f_{s_{i+1}} = 4f_{s_i} + 8p^2 x_{i+1} + 4p^2 - 8q^2 y_{i+1}}$ (IV)

Podobne rozważania dadzą nam wartości dla funkcji oceniającej w drugiej części okręgu (rys.9a).I tak przy wyborze $P_{i+1} = B = (x_{i+1}, y_{i+1}) = (x_i + 1, y_i - 1)$

$$f_{s_{i+1}} = F(x_{i+1} + \tfrac{1}{2}, y_{i+1} - 1) = f_{s_i} + 2p^2 x_{i+1} - 2q^2 y_{i+1} + q^2$$

lub inaczej $\boxed{4f_{s_{i+1}} = 4f_{s_i} + 8p^2 x_{i+1} - 8q^2 y_{i+1} + 4q^2}$ (V).

Zaś przy wyborze $P_{i+1} = C = (x_{i+1}, y_{i+1}) = (x_i, y_i - 1)$

$\boxed{4f_{s_{i+1}} = 4f_{s_i} - 8q^2 y_{i+1} + 4q^2}$ (VI)

Przed sformułowaniem algorytmu przypomnijmy,że pierwsza część okręgu,to te punkty okręgu,dla których spełniona będzie nierówność $p^2 x < q^2 y$.Druga zaczyna się wtedy gdy przestaje ona obowiązywać,a kończy gdy uzyskamy y<0 .Pozostaje jeszcze ustalić jak zmodyfikować funkcję oceniającą dla pierwszego piksla po przejściu z obszaru pierwszego do drugiego.

Okazuje się,że wystarczy przyjąć różnicę wartości funkcji oceniających,obowiązujących w poszczególnych obszarach przy wyborze punktu *B*:

$$f_{s+1} = F(x_i + \tfrac{1}{2}, y_i - 1) - F(x_i + 1, y_i - \tfrac{1}{2}) =$$
$$= p^2(-x_i + \tfrac{3}{4}) + q^2(-y_i - \tfrac{3}{4})$$

lub inaczej : $4f_{s_i} = p^2(-4x_i + 3) + q^2(-4y_i - 3) =$

$\boxed{-4(p^2 x_i + q^2 y_i) + 3(p^2 - q^2)}$ (VII)

Możemy teraz już sformułować procedurę rysującą okrąg *O[(a,b),r]* :

```
Procedure Okrag(a,b,r:integer;kol:byte);
var pp,qq,pp4,qq4,pp8,qq8,fx,fy,fs,x,y:longint;
   Procedure RysPkt(xo,yo,x,y:integer;k:word);
     begin
         { Rysuj punkt i symetryczne do niego wzgl.osi }
          Punkt(xo+x,yo+y,k);Punkt(xo-x,yo+y,k);
          Punkt(xo+x,yo-y,k);Punkt(xo-x,yo-y,k);
     end;
BEGIN
  x:=0;y:=r;
  pp:=sqr(x_aspekt);
  qq:=sqr(y_aspekt);
  pp4:=4*pp;
  qq4:=4*qq;
  pp8:=8*pp;
  qq8:=8*qq;
  fx:=0;
  fy:=qq8*r;fs:=pp4-qq4*r+qq;
```

```
  While fx<fy do          {punkty okręgu pierwszego obszaru}
    begin
      RysPkt(a,b,x,y,kol);
      Inc(x);
      fx:=fx+pp8;
      if fs<=0 then fs:=fs+fx+pp4
        else
          begin
            Dec(y);
            fy:=fy-qq8;
            fs:=fs+fx+pp4-fy;
          end;
    end;
    fs:=fs-((fx+fy) div 2)+3*(pp-qq);
    While y>=0 do           {punkty okręgu drugiego obszaru}
      begin
        RysPkt(a,b,x,y,kol);
        Dec(y);
        fy:=fy-qq8;
        if fs<=0 then
          begin
            Inc(x);
            fx:=fx+pp8;
            fs:=fs+fx-fy+qq4;
          end
           else fs:=fs-fy+qq4;
      end;
END;
```

Zmienne *x_aspekt* i *y_aspekt* są zmiennymi globalnymi.Przypomnijmy,że w naszym trybie *x_aspekt*=22 oraz *y_aspekt*=25.
Teraz odpowiednio zmieniając wielkość aspektu,a dokładniej jego mianownik *y_aspekt*,możemy rysować elipsy o osiach równoległych do osi OX i OY :

```
Procedure Elipsa(a,b,os1,os2:integer;kolor:byte);
begin
  y_aspekt:=Round(25*os1/os2);
  Okrag(a,b,os2,kolor);
  y_aspekt:=25;
end;
```

gdzie (*a,b*) to środek elipsy,zaś *os1* i *os2* to osie elipsy odpowiednio równoległe do osi OX i OY.Aby procedura *ELIPSA* dawała prawidłowy efekt,zmienne *x_aspekt* i *y_aspekt* powinny być zadeklarowane jako typu *LONGINT*.Ponadto procedury *OKRAG* i *ELIPSA* są wyposażone w mechanizm obcinania do brzegów ekranu,co gwarantuje odpowiednio skonstruowana wcześniej procedura *PUNKT*.

2.f. Wyprowadzanie tekstu na ekran

Wyprowadzanie tekstu na ekran zrealizujemy za pośrednictwem BIOSU,którego funkcje przerwania 10h działają również w trybie graficznym.
Znaki w trybie 13h mają wielkość 8x8,co daje dokładnie 25 wierszy i 40 kolumn ponumerowanych odpowiednio 0..24 i 0..39.Sformułujemy najpierw dwie procedury pomocnicze.Pierwsza będzie przesuwała kursor do wskazanego adresu na ekranie :

```
Procedure AT(kolumna,wiersz:byte);
var Rej :Registers;
```

```
begin
  With Rej do
    begin
      AH:=$02;
      BH:=0;
      DH:=wiersz;
      DL:=kolumna;
    end;
    INTR($10,Rej);
end;
```

Następna spowoduje wypisanie jednego znaku,w kolorze i na wybranym tle :

```
Procedure Pisz(asc, kol_z, kol_t :byte);
var rej:Registers;
begin
  With Rej do
    begin
      AH:=$0A;   AL:=asc;
      BH:=kol_t;    {kolor tła}
      BL:=kol_z;    { kolor znaku }
      CX:=1;        { ilość powtórzeń}
    end;
   INTR($10,Rej);
end;
```

I wreszcie procedura wyprowadzająca dowolny tekst o długości nie przekraczającej 255 znaków począwszy od adresu (*kolumna,wiersz*), w kolorze *kol_napisu* i na tle *kol_tla* :

```
Procedure Wpisz(kolumna,wiersz,kol_napisu,kol_tla:byte;zd:string);
var i:byte;
begin
  for i:=1 to length(zd) do
    begin
      AT(kolumna+i-1,wiersz);
      Pisz(Ord(zd[i]),kol_napisu,kol_tla);
    end;
end;
```

Do wyprowadzania liczb na ekran można posłużyć się już procedurą **Wpisz** :

```
Procedure PiszLczbeCalk(kolumna,wiersz ,kol_napisu,kol_tla:byte;liczba :integer);
var zd :string;
begin
    Str(liczba,zd);
   Wpisz(kolumna,wiersz,kol_napisu,kol_tla,zd);
end;
```

W zbliżony sposób wypiszemy również dowolną liczbę rzeczywistą.Wczytywanie danych w zasadzie należy samodzielnie zrealizować,co prawda polecenia

AT(kol,wier) ;readln(zmienna);

spełniają zadanie,akcja widoczna jest na ekranie,lecz brak charakterystycznego,migającego kursora każe szukać lepszego rozwiązania, tzn procedurę wczytującą konstruować samodzielnie.Zarówno w trybie tekstowym, jak i graficznym istnieje możliwość przewijania zawartości prostokątnego okna o wielokrotność wysokości wiersza w górę lub w dół.Przy przewijaniu w górę,pewna liczba wierszy zniknie z okna, licząc od góry i jednocześnie dokładnie taka sama liczba wierszy na dole okna wypełniona zostanie spacjami z podanym atrybutem.Zjawisko to nastąpi niezależnie, czy w oknie znajduje się tekst czy grafika.

Podajemy uniwersalną procedurę,która przewija w górę lub w dół zawartość okna :

```
Procedure Skroluj(kg,wg,kd,wd,ile,kolor: byte);
var rej:Registers;
begin
  With rej do
    begin
      if ile>0 then AH:=$06 else AH:=$07;
      AL:=abs(ile); { jeśli ile>0 przewijanie do góry wpp w dół o |ile| }
      BH:=kolor;    { kolor wypełnienia pustego wiersza }
      CH:=wg;
      CL:=kg;
      DH:=wd;
      DL:=kd;
    end;
    INTR($10,rej);
end;
```

gdzie (kg,wg) to współrzędne lewego górnego,(kd,wd) współrzędne prawego rogu przewijanego okna,liczone tak jak dla trybu tekstowego.Dla parametru *ile*>0 przewijamy zawartość okna do góry zaś dla *ile*<0 w dół o |*ile*| wierszy czyli o 8x*ile* linii ekranu.Na koniec tego rozdziału warto wspomnieć o możliwości wyświetlania innych znaków niż standardowe.Najprościej chyba jest przedefiniować znaki używane przez BIOS.W tym celu musimy przygotować tablicę definicji 256 znaków,na każdą definicję tyle bajtów, jaką wysokość znaku chcemy uzyskać.Dla trybu 13h jako typowe stosuje się znaki o wysokości 8 linii (8x8) w 25 wierszach lub 14 linii (8x14) w 14 wierszach.Możliwa jest również inna kombinacja wierszy i linii,o ile iloczyn wierszy i linii nie przekroczy liczby 200.Załóżmy,że dysponujemy definicją 256 znaków 8x8 dla 25 wierszy po 8 linii zapisaną w pliku.Procedura **Definiuj_Fonty** wczyta nowe definicje i zamieni standardowe znaki BIOSU,gdzie parametr *zbior* jest nazwą zbioru,który zawiera nowe definicje:

```
var font:array[byte,1..8] of byte;
Procedure Definiuj_Fonty(zbior :string);
var plik :file;
      rej:registers;
begin
  Assign(plik,zbior);
  Reset(plik,1);
  BlockRead(plik,font,2048);
  Close(plik);
  rej.AH:=$11;
  rej.AL:=$0021;
  rej.ES:=Seg(font);
  rej.BP:=Ofs(font);
  rej.CX:=8;                { liczba linii na znak }
  rej.BL:=$02;              { 25 wierszy }
  INTR($10,regs)
end;
```

Przy innej wielkości wierszy,w rejestrze BL podajemy :

$01 dla 14 wierszy
$03 dla 43 wierszy
$00 a potem dodatkowo w rejestrze DL liczba wierszy,w innym przypadku.

Przy tej samej okazji możemy oczywiście rozwiązać problem polskich liter,po prostu w przygotowanym pliku uwzględnimy definicję polskich znaków diaktrytycznych.

2.g. Zmiana kolorów palety

W trybie 13h na ekranie może być widocznych jednocześnie 256 kolorów.Kolor każdego piksla opisany jest przez osiem bitów.Ośmiobitowy numer koloru wskazuje bezpośrednio na jeden z 256 rejestrów DAC (*Video Digital to Analog Converter*),który zawiera pełną,18-bitową definicję koloru każdego koloru.
Kolor opisany jest przez trzy składowe *R*(czerwona),*G*(zielona),*B*(niebieska) i tworzony jest przez zsumowanie tych trzech składowych,z których każda opisana jest przez 6 bitów (a więc przyjmuje wartości z zakresu 0-63).Teoretycznie zatem można wygenerować 64^3=262144 barwy.Zmiana danego koloru polega na zmianie jego składowych R,G i B.
Przy pomocy przerwania 10h BIOSU zadanie to zrealizuje procedura :

```
Procedure RGB_(kolor,r,g,b:byte);
var rej :registers;
begin
  With rej do
    begin
      AH:=$0010;
      AL:=$0010;
      BX:=kolor; {zmieniany kolor a właściwie numer rejestru DAC 0-255}
      DH:=r;
      CH:=g;
      CL:=b
    end;
    INTR($10,rej);
end;
```

Można również odczytać wartości R,G i B danego koloru,co nieraz może być przydatne :

```
Procedure Jakie_RGB_(kolor:byte;var r,g,b:byte);
var rej :registers;
begin
  With rej do
    begin
      AH:=$0010; AL:=$0015;
      BX:=kolor;
    end;
    INTR($10,rej);
  With rej do
    begin
      r:=DH;  g:=CH;  b:=CL;
    end;
end;
```

Istnieje jednak możliwość zmiany palety kolorów bez pośrednictwa BIOS,mianowicie poprzez bezpośrednie wpisanie do portów wejścia/wyjścia nowych zawartości rejestrów DAC i tak :

```
Procedure RGB(kolor,r,g,b:byte);
begin
 Port[$3C8]:=kolor;
 Port[$3C9]:=r;
 Port[$3C9]:=g;
 Port[$3C9]:=b;
end;
```

Natomiast odczyt aktualnych wartości R,G i B można zrealizować tak :

```
Procedure Jakie_RGB(kolor:byte;var r,g,b:byte);
begin
  Port[$3C7]:=kolor;          { kolor do odczytu }
  r:=Port[$3C9];
  g:=Port[$3C9];
  b:=Port[$3C9];
end;
```

Proponujemy wypróbować i porównać szybkość działania procedur RGB_ i RGB w następ-pującej sytuacji :

```
Program Badaj_Palete;
uses crt,graf_13;

Procedure Rysuj_Kreski;
var i:integer;
begin
    for i:=1 to 252 do Linia(i,10,i,180,i);
end;

Procedure Rob_Palete;
var i:integer;
begin
  for i:=1 to 63 do
    begin
      RGB(i,0,0,i);             { zrób odcienie niebieskigo}
      RGB(i+63,0,i,0);          { zrób odcienie zielonego }
      RGB(i+126,i,0,0);         { zrób odcienie czerwonego }
      RGB(i+189,i,i,i);         { zrób odcienie szarości }
    end;
end;

BEGIN
  Grafika;
  Rysuj_Kreski;
  Repeat Until KeyPressed;
  Rob_Palete;
  Repeat Until KeyPressed;
  TextMode(3);
END.
```

Zakładamy,że procedury *Grafika,Linia,RGB* zawarte są w module *GRAF_13*.Jeśli w procedurze *Rob_Palete*,zmienimy procedurę *RGB* procedurą *RGB_*,dynamika akcji znacznie się pogorszy.Mimo,że procedura *RGB* działa o wiele szybciej niż *RGB_*,nie jest całkowicie pozbawiona wad.W naszym programie paleta kolorów była zmieniona tylko raz.Wyobraźmy sobie,że po wykonaniu pewnego rysunku dokonujemy wielokrotnej zmiany kolorów.W takim wypadku dynamika akcji oczywiście zostanie zachowana,jednak na ekranie może pojawić się nieprzyjemne "śnieżenie" co popsuje cały efekt.Można temu zaradzić poprzedzając zmianę kolorów wspominaną już procedurą *SYNCHRONIZACJA*.
Na dyskietce demonstracyjnej,program WALCE.EXE pokazuje proste wykorzystanie zmiany palety,w celu uzyskania efektu obrotu w przestrzeni.

2.h. Zapamiętanie i odtwarzanie ekranu

Omówimy z kolei zpamiętanie i odtworzenie całego lub części ekranu,co dla technik animacyjnych jest zwykle podstawowym pomysłem.

Jak pamiętamy,zmienna *scr:array [0..199,0..319] absolute $A000:0*,służy do reprezentacji wyglądu ekranu w pamięci komputera.Wszelkie więc operacje na pamięci ekranu lub jego części,będą dokonywane na tej tablicy.Zapamiętać aktualny wygląd ekranu znaczy tyle,co przesłać zawartość tablicy *SCR* na inną zmienną.Odtworzyć wygląd ekranu oznacza przesłanie z tej innej zmiennej na zmienną *SCR* odpowiedniego obszaru pamięci.Najrozsądniej będzie w tej sytuacji posłużyć się zmienną typu *pointer*.Oto procedura zapamiętująca cały ekran :

```
Procedure ZapamietajEkran(var pr:pointer);
begin
  GetMem(pr,64000);
  Move(scr,pr^,64000)
end;
```

oraz procedura odtwarzająca wygląd całego ekranu:

```
Procedure OdtworzEkran(pr:pointer);
begin
   Move(pr^,scr,64000);
end;
```

I dla kompletu procedura zwalniająca niepotrzebny obszar pamięci :

```
Procedure ZwolnijEkran(pr:pointer);
begin
  FreeMem(pr,64000);
end;
```

Oto krótki program demonstrujący zastosowanie ww procedur w prostej animacji :

```
Program pokaz1;
uses crt,graf_13;
Procedure Kwadrat(x,y,bok,kolor :integer);
begin
   Linia(x,y,x+bok,y,kolor);
   Linia(x+bok,y,x+bok,y+bok,kolor);
   Linia(x+bok,y+bok,x,y+bok,kolor);
   Linia(x,y+bok,x,y,kolor);
end;

Procedure PrzygotujTlo;
var i:integer;
begin
    Tlo(9);
    for i:=0 to 199 do Linia(0,10,319,i,i);
    Wpisz(1,12,14,9,'Animacja kwadratu');
end;
Procedure Pokaz;
var x,dx,bok,db :integer;
       stary :pointer;
begin
  x:=50;dx:=4;
  bok:=40;db:=-1;
  ZapamietajEkran(stary);
  Repeat
     PrzypomnijEkran(stary);    { Odtwórz poprzednie tło }
     Synchronizacja;
     Kwadrat(x,100,bok,66);     { narysuj nową fazę }
     Inc(x,dx);                 {przygotuj nową fazę }
```

```
    if ((x>300) or (x<1)) then dx:= -dx;
    Inc(bok,db);
    if ((bok>40) or (bok<-40)) then db:= -db;
  Until KeyPressed;
  ZwolnijEkran(stary);
end;

BEGIN
  Grafika;
  PrzygotujTlo;
  Pokaz;
  TextMode(3);
END.
```

Wywołanie w procedurze POKAZ,*synchronizacji*,co prawda trochę zwalnia akcję,ale uspakaja ekran,a dokładniej proces rysowania kwadratu.Ponadto zakładamy,że*Grafika,Linia,Tlo, Synchronizacja,Wpisz ,ZapamietajEkran,PrzypomnijEkran* i *ZwolnijEkran* są zawarte w module **GRAF_13**.Zaprezentowana technika animacji polega na odtworzeniu tła przed wykonaniem rysunku kwadratu w nowym położeniu.Mimo,że operujemy na dość dużym bloku pamięci,animacja wykonuje się w przyzwoitym tempie.W dalszej części skonstruujemy procedurę zapamiętującą tę część ekranu,która w danym momencie jest potrzebna.Nieraz może zdarzyć się,że wykonanie rysunku wymaga pracochłonnych obliczeń,lub z innych względów trwa długo.Rozsądnie jest zatem tak uzyskaną grafikę zapisać na dysku,by kiedyś skorzystać z jej usług w nowej sytuacji.Grafika zwykle zajmuje sporo miejsca na dysku (w naszym przypadku 64 KB),ale zawsze przychodzi zapłacić jakąś cenę.Oto następne procedury :zapisująca zawartość ekranu na dysk i wczytująca do pamięci ekranu zapisany wcześniej wygląd ekranu:

```
Procedure ZapiszEkran(nazwa:string);
var plik:file of Ekran;
begin
  Assign(plik,nazwa);
  Rewrite(plik);
  write(plik,scr);
  Close(plik);
end;

Procedure WgrajEkran(nazwa:string);
var plik:file of Ekran;
begin
  Assign(plik,nazwa);
  Reset(plik);
  read(plik,scr);
  Close(plik);
end;
```

Należy zaznaczyć,że nie jest to jedyny sposób zapisu i odczytu zawartości ekranu,ale niezwykle prosty,korzysta bowiem z globalnej zmiennej *SCR*,która "pamięta" zawsze bieżący wygląd ekranu.Ponadto po wczytaniu zawartość pojawia się natychmiast na ekranie.
W programie demonstracyjnym *POWIERZ.EXE* demonstrowane są obie ostatnie procedury w animacji powierzchni funkcyjnej.Do tej pory operowaliśmy na pamięci całego ekranu.W wielu jednak sytuacjach potrzeba i wystarcza zapamiętać lub odtworzyć jego część.Zacznijmy jednak od czegoś innego,mianowicie od narysowania prostokątnego obszaru zamalowanego jednolicie zadanym kolorem.Co prawda niewiele to ma wspólnego z naszym zamiarem zapamiętania części ekranu,ale pokaże sposób postępowania z prostokątną częścią ekranu.

Aby w miarę szybko zamalować prostokąt o wymiarach *a* i *b*,najpierw narysujemy pierwszą, górną linię wnętrza prostokąta,po czym dokonamy *b-1* jej kopii w wierszach od 2 do *b* :

```
Procedure Bar_(x1,y1,x2,y2:integer;kolor:byte);
var i,dl,il:integer;
begin
   dl:=x2-x1;                { określ długość linii prostokąta }
   if x1+dl>319 then dl:=319-x1;
   il:=y2-y1;               {określ ilość linii prostokąta do wypełnienia }
   if y1+il>199 then il:=199-y1;
   FillChar(src[y1,x1],dl,kolor);              { narysuj pierwszy wiersz }
   for i:=y1+1 to y1+il-1 do                   { wykonaj kolejne kopie  }
        Move(scr[y1,x1],scr[i,x1],dl);
       {prześlij wiersz pierwszy do pamięci i-ego }
end;
```

Aby z procedury można było bezpiecznie korzystać niezbędne są jeszcze zabezpieczenia,w sytuacji źle podanych współrzędnych lewego górnego (x_1 , y_1) i prawego dolnego wierzchołka (x_2 , y_2),które należałoby umieścić bezpośrednio za "beginem" a przed obliczeniem długości linii i ich liczby do wypełniania:

```
   if x1<0 then x1:=0;
   if x1>319 then x1:=319;
   if y1<0 then y1:=0;
   if y1>199 then y1:=199;
   if ((x2<x1) or (y2<y1)) then Exit;
```

Podobny mechanizm zastosujemy zapamiętując pewną,prostokątną część ekranu.Zmienną,na którą zapamietamy zawartość prostokąta będziemy tworzyli tak jak w przypadku zamalowania,linia po linii.Zawartość każdej linii zapamiętywanego obszaru prześlemy do odpowiedniego dla danej linii,adresu zmiennej ***PR***.

```
Procedure Zapamietaj(a,b,dl,sz : integer; var pr : pointer);
var i:integer;
begin
   dl:=dl+1;
   if a+dl>319 then dl:=319-a;       {czy prostokąt mieści się na ekranie }
   if b+sz>199 then sz:=199-b;
  GetMem(pr,dl*sz);          { zarezerwuj odpowiedni blok pamięci }
  for i:=0 to sz-1 do        {przesyłaj zawartość linii w odp.adres pr^}
      Move(scr[b+i,a],PTR(Seg(pr^),Ofs(pr^)+dl*i)^,dl);
end;
```

Przypomnienie zawartości prostokątnego obszaru ekranu :

```
Procedure Przypomnij(a,b,dl,sz : integer; pr : pointer);
var i : integer;
begin
   dl:=dl+1;
   if a+dl>319 then dl:=319-a;
   if b+sz>199 then sz:=199-b;
   for i:=0 to sz-1 do       {przesyłaj zawartość odp.linii na ekran}
      Move(PTR(Seg(pr^),Ofs(pr^)+dl*i)^,scr[b+i,a],dl);
end;
```

Niepotrzebną pamięć zwalnia procedura :

```
Procedure Zwolnij(a,b,dl,sz : integer; pr : pointer);
```

```
begin
   dl:=dl+1;
   if a+dl>319 then dl:=319-a;
   if b+sz>199 then sz:=199-b;
  FreeMem(pr,dl*sz)
end;
```

gdzie (*a,b*) to współrzędne lewego,górnego wierzchołka okienka,*dl* i *sz* to odpowiednio długość i szerokość zapamiętywanego (odtwarzanego) okienka.

2.i. Organizacja dodatkowego ekranu

W prezentowanym wyżej programie *Pokaz1* pokazaliśmy prostą animację kwadratu poruszającego się po ekranie.Aby uzyskać efekt "nienaruszonego" tła zapamiętana zawartość całego ekranu była odtwarzana przed wykonaniem rysunku kwadratu w nowym położeniu. Kwadrat jest obiektem mało pracochłonnym,zawartość ekranu przypominana była szybko i w rezultacie wygląda to dość przyzwoicie.Gorzej jest jeśli dysponujemy wolnym komputerem lub nową fazę animacji tworzy skomplikowany rysunek.Wtedy mimo szybkiego odtworzenia tła widać cały proces powstawania rysunku w nowym położeniu i efekt jest średni.
W takiej sytuacji rysowanie należy ukryć przed obserwatorem a pokazać jedynie rezultat końcowy.Tak prowadzona animacja zapewni niezbędną płynność akcji i elegancję pokazu.
Jak już wspominaliśmy w trybie 13h,BIOS udostępnia nam tylko jedną stronę. Powtórzmy,że ekran zajmuje 64 KB,ale włączając tryb 13h BIOS włącza jednocześnie tzw tryb adresowania modulo 4.Uzyskujemy bardzo wygodny sposób adresownia pamięci ekranu (liniowy),niestety kosztem wykorzystania całej przestrzeni adresowej karty. Dla nas ekran "widoczny" to po prostu tablica *SCR* adresowana bespośrednio do pamięci ekranu.Ekranem "niewidocznym",który za chwilę stworzymy,będzie zmienna o adresie innym niż adres zmiennej *SCR* (przypomnijmy ten adres: $A000:0).Potrzebna jest na to oczywiście dodatkowa pamięć.Wyjścia są dwa.
Po pierwsze rolę ekranu "niewidocznego" może pełnić zmienna *strona*:pointer.
Tego typu zmiennej możemy w dowolnej chwili przydzielić pamięć oraz ją zwolnić.Nauczmy się "rysować" na takim ekranie. Oczywiście kluczową sprawą to narysowanie punktu,a narysowanie punktu to po prostu wpisanie do odpowiedniego adresu nowego ekranu koloru punktu.
Niech zmienne ***segment,ofset*** :word; będą zmiennymi globalnymi.
Utworzenie nowego ekranu to przydzielenie zmiennej *strona* pamięci oraz obliczenie adresu tej zmiennej,który przekażemy zmiennym *segment* i *ofset* :

```
  GetMem(strona,64000);
  segment:=Seg(strona^);
  ofset:=Ofs(strona^);
```

Czyszczenie nowego ekranu : **FillChar(strona^,64000,kolor);**
Rysowanie punktu zapewni procedura:

```
Procedure Plot(x,y:integer;kolor:byte);
begin
  if ((x>=0) and (x<320) and (y>=0) and (y<200)) then
     Mem[segm:ofset+320*y+x]:=kolor;
end;
```

Po wykonaniu rysunku trzeba oczywiście pokazać "niewidoczny" ekran :

```
Procedure PokazEkran;
```

```
begin
  Move(strona^,scr,64000);
end;
```

Inne czynności na tym ekranie wykonujemy podobnie jak dla ekranu "widocznego" np wgranie z dysku zapamiętanego rysunku :

```
  Procedure WgrajObraz(nazwa :string);
  var plik :file;
  begin
      Assign(plik,nazwa);
      Reset(plik,1);
      BlockRead(plik,strona^,64000);
      Close(plik);
  end;
```

Kończąc program należy pamiętać o zwolnieniu pamięci naszej dodatkowej strony: **FreeMem(strona,6400);.**
Po drugie wreszcie,tworząc dodatkową stronę możemy posłużyć się również zmienną typu *Ekran*,który to typ był już definiowany dla obsługi grafiki.Przypomnijmy zdefiniowany typ :

```
  type Ekran=array [0..199,0..319] of byte;
```

W takim przypadku utworzenie dodatkowej strony to deklaracja zmiennej globalnej :

```
   var :strona : ^Ekran;
```

oraz zarezerwowanie pamięci dla tej zmiennej :

```
  GetMem(strona,64000); .
```

Rysowanie punktu :

```
       Procedure Pkt(x,y:integer;kolor:byte);
       begin
         if ((x>=0) and (x<320) and (y>=0) and (y<200)) then
             strona^[y,x]:=kolor;
       end;
```

zaś pokazanie rysunku :

```
       Procedure PokazEkran;
       begin
         Move(strona^,scr,64000);
       end;
```

Oto przykładowy program demonstrujący wykorzystanie dodatkowej strony w animacji,gdzie w module **graf_13** zawarte są procedury *Grafika,RGB* oraz *Czekaj* :

```
Program D_strony;
uses crt,graf_13;
type Ekran=array [0..199,0..319] of byte;
var strona:^Ekran;                { będzie użyty dodatkowy ekran }

Procedure Pkt(x,y:integer;kolor:byte);
begin
  if ((x>=0) and (x<320) and (y>=0) and (y<200)) then
  strona^[y,x]:=kolor;
```

```
end;
{-------------------------------------------}
Procedure PokazEkran;
begin
  Move(strona^,scr,64000);
end;
{-------------------------------------------}
Procedure ZrobPalete;
var i:integer;
begin
 for i:=1 to 32 do RGB(i,45,32+i,20+i);
 for i:=33 to 96 do RGB(i,40,60,i-32);
 for i:=97 to 160 do RGB(i,i-96,60,i-32);
 for i:=161 to 224 do RGB(i,40,i-160,60);
end;
{-------------------------------------------}
Procedure Kreska;
var i:integer;
begin
  for i:=1 to 20 do Pkt(160,90+i,250);
end;
{-------------------------------------------}
Procedure Spirala(fi:real;kol:byte);
var alfa:real;
       x,y,r:integer;
       kolor:byte;
begin
  alfa:=0;
  kolor:=kol;
   r:=3;
  Repeat
    x:=Round(160+0.5*r*cos(0.6*alfa+fi));
    y:=Round(100-0.2*r*sin(0.3*alfa+fi));
    alfa:=alfa+0.02;
    Inc(kolor);
    Inc(r);
    Pkt(x,y,kolor);
  Until alfa>2*Pi;
end;
{--------------------------------------}
Procedure Demo;
var kolor:byte;
       fi:real;
begin
   fi:=0;kolor:=34;
   Repeat
     FillChar(strona^,64000,200);        { Wyczyść ekran niewidoczny }
     Kreska;                             { wykonaj rysunek }
     Spirala(fi,kolor);
     PokazEkran;                         { pokaż co zrobiłeś }
     Dec(kolor,2);
     if kolor<1 then kolor:=224;         { przygotuj następną fazę }
     fi:=fi+0.2;
   Until KeyPressed;
end;
BEGIN
  GetMem(strona,64000);                 { Przygotuj dodatkowy ekran }
  ZrobPalete;
  Demo;
  Czekaj;
  FreeMem(strona,64000);                { Zwolnij dodatkowy ekran }
```

```
  TextMode(3);
END.
```

2.j. Prosta procedura wyboru

Jeśli w programie realizujemy pewien scenariusz wygodnie jest dysponować procedurą wybierającą z listy żądaną czynność.Wyobraźmy sobie,że wypisaliśmy w pewnym miejscu ekranu listę oferowanych możliwości programu.Następnie chcemy "wędrować" po napisach,do momentu podjęcia decyzji.W tym celu sformułujemy procedurę zmieniającą koloryt określonego prostokątnego obszaru,ale tak by napisy (lub inne obiekty) były nadal widoczne tzn ten obszar pokryjemy "cieniem" określonego koloru.Napisy lub rysunek tworzone są w określonym kolorze na określonym tle,dla nas w tym wypadku istotny będzie kolor tła.
Proponujemy procedurę zmieniającą tło zmodyfikować następująco :

```
Procedure Tlo(kolor:byte);
begin
  FillChar(scr,200*320,kolor);
  tlo_:=kolor;
end;
```

gdzie zmienna ***tlo_***,będzie zmienną globalną pamiętającą kolor aktualnego tła ekranu.
Zmienić koloryt pewnego prostokątnego obszaru wg naszego zamysłu,znaczy tyle co na każdym kolorze punktu tego obszaru wykonać operację :
(*kolor_Punktu* ***XOR*** *kolor_tła)* ***XOR*** *kolor_po_zmianie* .
Oto procedura realizująca to zadanie :

```
Procedure Cien(x,y,dl,sz:integer;kolor:byte);
var i,j:integer;
begin
  for i:=1 to sz do
    for j:=1 to dl do
        scr[y+i-1,x+j-1]:=((scr[y+i-1,x+j-1] xor tlo_) xor kolor);
end;
```

A teraz prosty program pokazujący zastosowanie w/w procedury :

```
Program Wybor_z_menu;
uses crt,graf_13;
var nr:integer;
    zap:pointer;
Procedure Cien(x,y,dl,sz:integer;kolor:byte);
var i,j:integer;
begin
  for i:=1 to sz do
    for j:=1 to dl do
        scr[y+i-1,x+j-1]:=((scr[y+i-1,x+j-1] xor tlo_) xor kolor);
end;

Procedure Wybor(x,y,dl,ilosc :integer;kolor,na_tle:byte;var nmr:integer);
var ch :char;                    { x,y- współrzędne pierwszego obszaru}
    y1,y1s :integer;
    jakie_tlo:byte;
begin
   nmr:=1;
   jakie_tlo:=tlo_;
   tlo_:=na_tle;
   y1:=8*y;
```

```
   y1s:=y1;
   Cien(x,y1,dl,8,kolor);      {zmień koloryt pierwszego napisu }
  Repeat
     Repeat
       ch:=ReadKey;
     Until ch in [#13,#27,#72,#80];
   Case ch of                     { podejmij dalszą decyzję }
     #80 :begin Inc(y1,8);Inc(nmr);end;
     #72 :begin Dec(y1,8);Dec(nmr);end;
     #13,#27 :begin;
               Cien(x,y1,dl,8,kolor);
               if ch=#27 then nmr:=0;
               tlo_:=jakie_tlo;
               Exit;
             end;
   end;
   if nmr>ilosc then nmr:=1;            { sprawdź poprawność }
   if nmr<1 then nmr:=ilosc;
   if y1s<>y1 then
     begin
      y1:=8*(nmr-1+y);
      Cien(x,y1s,dl,8,kolor);          { zlikwiduj stare podświetlenie }
      Cien(x,y1,dl,8,kolor);           { podświetl nowy obszar }
      y1s:=y1;
     end;
 Until ord(ch)=13;
 end;
Procedure Menu;
begin
  Tlo(9);
  Linia(0,0,320,199,12);
  Zapamietaj(35,35,70,55,zap);  { Zapamiętaj tło pod ramką }
  Bar_(35,35,100,85,58);        { Narysuj zamalowany prostokąt z "cieniem" }
  Cien(40,85,66,5,1);           { Zrób cień okienka z napisami }
  Cien(101,40,5,45,1);
  Wpisz(5,5,255,58,'Opcja 1');  { wykonaj napisy }
  Wpisz(5,6,255,58,'Opcja 2');
  Wpisz(5,7,255,58,'Opcja 3');
  Wpisz(5,8,255,58,'Opcja 4');
  Wpisz(5,9,255,58,'Wyjście');
end;
{-------------------------------------------}
BEGIN
  Grafika;
  Menu;
  Wybor(40,5,56,5,66,58,nr);
 {Dokonaj wyboru,na zmienną nr zapamiętaj decyzję }
  Przypomnij(35,35,70,55,zap);  { Odtwórz tło pod ramką menu }
  Zwolnij(35,35,70,55,zap);     { Zwolnij pamięć zmiennej ZAP }
  Czekaj;                       { Czekaj na klawisz ESC }
  TextMode(3);
END.
```

Zakładamy,że w module **Graf_13** znajdują się wszystkie potrzebne procedury do uruchomienia tego przykładu.

2.k. Obsługa myszy w trybie 13h

Co do wygody jaką mamy używając mysz,nie należy nikogo przekonywać.Obsługa programu przy pomocy myszy,należy do standardowego rozwiązania większości sytuacji jakie mogą zdarzyć się na ekranie.
Jeżeli program ma korzystać z myszy,to oprócz fizycznego dołączenia do komputera,niezbędne jest jeszcze załadowanie do pamięci odpowiedniego programu obsługi.Zainstalowana w ten sposób mysz może być używana w trybie tekstowym jak i w graficznym (z różnym wyglądem kursora myszy). W trybach graficznych VGALo,VGAMed,VGAHi (pod kontrolą sterownika egavga.bgi) użycie i sterowanie myszą można uzyskać wykorzystując przerwanie nr 51 (33h) procesora wywołując odpowiednio procedurę ***INTR*** .Zasada użycia w programie pascalowym jest następująca:

```
Procedure Korzystam_z_przerwania_51;
var rej:Registers;
begin
  rej.AX:=nr_funkcji_przerwania;
    {innym rejestrom przypisać wymagane wartości}
  INTR($33,rej);
end;
```

A oto krótki przegląd standardowych funkcji usługowych przerwania 51 :

Nr fun	Realizowana czynność
0	- inicjacja myszy
1	- uwidocznienie kursora myszy
2	- ukrycie kursora myszy
3	- odczyt bieżącego położenia i statusu myszy
4	- ustalenie wyjściowego położenia kursora myszy
5	- dostarczenie danych o wciśniętych przyciskach
6	- dostarczenie danych o zwolnionych przyciskach
7	- definiowanie graficznych współrzędnych w osi X
8	- definiowanie graficznych współrzędnych w osi Y
9	- definiowanie postaci kursora graficznego
10	- definiowanie postaci kursora tekstowego
11	- odczyt liczników przemieszczeń myszki
15	- wybór współczynnika czułości myszki
16	- definiowanie martwych stref ekranu

Na przykład chcąc przedefiniować wygląd kursora myszy w trybie graficznym,należy przygotować definicję nowego kursora (o sposobie definiowania będzie jeszcze mowa w tym rozdziale) i uruchomić następującą procedurę :

```
Procedure Def_M(rodzaj:integer);
type Kurs =array [1..32] of word;
const Lapka:Kurs
      =($F3FF,$E1FF,$E1FF,$E1FF,$E1FF,$E00F,$E001,$8000,
        $0000,$0000,$0000,$0000,$8001,$C001,$E003,$F007,
        $0000,$0C00,$0C00,$0C00,$0C00,$0C00,$0DB0,$0DB6,
        $6FFE,$6FFE,$6FFE,$7FFE,$3FFC,$1C1C,$0FF8,$0000);
     Strzalka:Kurs
      =($9FFF,$8FFF,$87FF,$83FF,$81FF,$80FF,$807F,$803F,
      $801F,$800F,$80FF,$887F,$987F,$fC3F,$fC3F,$fE3F,
      $0000,$2000,$3000,$3800,$3C00,$3E00,$3F00,$3F80,
      $3FC0,$3E00,$3600,$2300,$0300,$0180,$0180,$0000);
var      Rej:registers;
```

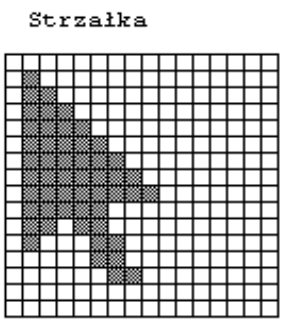
```
            kursor_myszy:kurs;
begin
  Rej.AX:=9;
  Rej.BX:=1;     {w rejestrze BX i CX podajemy współrzędne celownika }
  Rej.CX:=1;
  if rodzaj=1 then kursor_myszy:=strzalka
                 else kursor_myszy:=lapka;
        Rej.ES:=Seg(kursor_myszy);
        Rej.DX:=Ofs(kursor_myszy);
     INTR($33,Rej);
end;
```

Kłopot w tym,że poza trybem VGALo,VGAMed i VGAHi (pod kontrolą sterownika egavga.bgi) wystąpią problemy z graficznym kursorem i użycie myszy w zasadzie jest niemożliwe.Poza tym pracując w trybie wielokolorowym można pokusić się o wykonanie kursora myszy w każdym z możliwych kolorów.Reasumując,przy konstrukcji procedur obsługi myszy dla nietypowych trybów,w tym i trybu 13h,będziemy unikali funkcji nr 1,2,9.Czynności realizowane przez te funkcje spróbujemy zdefiniować sami.Pozostałe zaś funkcje,które nie odwołują się do wyglądu kursora myszy oczywiście wykorzystamy.Zdefiniujemy najpierw funkcje : testującą obecność myszy,stan przycisków oraz procedury podające położenie myszy, stan przycisków,ustalające dostępny obszar dla kursora oraz przesuwającą mysz do punktu (x,y) :

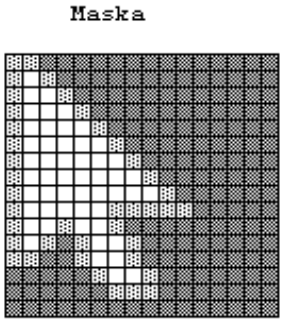
```
Function Mysz_O:boolean;    { Czy mysz jest zainstalowana ? }
var rej:Registers;
begin
  rej.AX:=0;
  Intr($33,rej);
  if rej.AX=0 then Mysz_O:=false
    else Mysz_O:=true;
end;

Function WPrzycisk(nr:integer):boolean;
 { Czy przycisk nr jest wciśnięty }
var rej :Registers;
begin
  rej.AX:=3;
  Intr($33,rej);
  if (rej.BX and (1 shl(nr-1)))=0
    then WPrzycisk:=false
      else WPrzycisk:=true;
end;

Procedure Pozycja_M(var x,y,przyciski:integer);
var rej:Registers;
begin  {Podaj aktualne położenie myszy i nr wciśniętego przycisku }
  rej.AX:=3;
  Intr($33,rej);
  x:=rej.CX;
  y:=rej.DX;
  przyciski:=rej.BX;
end;

Procedure Granice_M(x_min,x_max,y_min,y_max:integer);
 { Ustal obszar dostępny dla myszy }
var rej:Registers;
begin
  rej.AX:=7;
  rej.CX:=x_min;
  rej.DX:=x_max;
```

```
  Intr($33,rej);
  rej.AX:=8;
  rej.CX:=y_min;
  rej.DX:=y_max;
  Intr($33,rej);
end;

Procedure Ustaw_M(x,y:integer);   { Przesuń mysz do punktu (x,y) }
var rej:Registers;
begin
  rej.AX:=4;
  rej.CX:=x;
  rej.DX:=y;
  Intr($33,rej);
end;
```

Pozostaje teraz wykonać kursor myszy i zmusić do współpracy z w/w procedurami.Najpierw zdefiniujemy kursor myszy.W naszej sytuacji sposób definiowania jest w zasadzie dowolny,pod warunkiem że potrafimy go potem sensownie narysować i animować.Posłużymy się jednak metodą jaką firmowe programy obsługi przechowują standardowy kursor graficzny,tak by można było swobodnie korzystać z ewentualnych zbiorów definicji wykonanych dla innych trybów.Standardowa definicja,to tablica (lub dwie) opisująca kursor jaki można narysować w siatce 16x16 punktów.



rys.10



rys.10a

Przypuśćmy,że chcemy zdefiniować kursor w postaci strzałki takiej jak na rysunku 10.Definicja składa się z definicji strzałki (*rys 10*) i jej maski (*rys 10a*).Obie defininicje można zgromadzić w jednej tablicy tak jak jest to uczynione w cytowanej wyżej procedurze **Def_M**.Wtedy tablica w 32 elementach ma zapisany wygląd 16 linii maski i 16 linii dla strzałki. Definicja strzałki określa który z punktów obszaru 16x16,przeznaczonego na całą mysz,ma być narysowany zaś definicja maski w jaki sposób mysz przykryje tło.Chodzi na ogół o to by część obszaru 16x16,poza strzałką była "przeźroczysta".Aby to uzyskać poddamy operacji AND (iloczyn logiczny) maskę z zawartością ekranu a następnie otrzymany rezultat poddamy operacji XOR (różnicy symetrycznej) z wyglądem samej strzałki.Stan punktów na ekranie będzie więc ustalany wg zasady :

Strzałka	Maska	Ekran
0	0	punkt w kolorze tła
1	0	punkt koloru strzałki
0	1	bez zmiany
1	1	inwersja koloru

Wróćmy teraz do defincji strzałki (*rys.10*).Niech zamalowane punkty siatki reprezentuje wartość 1,pozostałe wartość 0.Czwarta linia definicji strzałki (licząc od góry) jest zatem opisana : binarnie **0111000000000000** ,

dziesiętnie $1\cdot16384+1\cdot8192+1\cdot4096+0\cdot2048+\ldots0\cdot2+0\cdot1=$**28672** ,

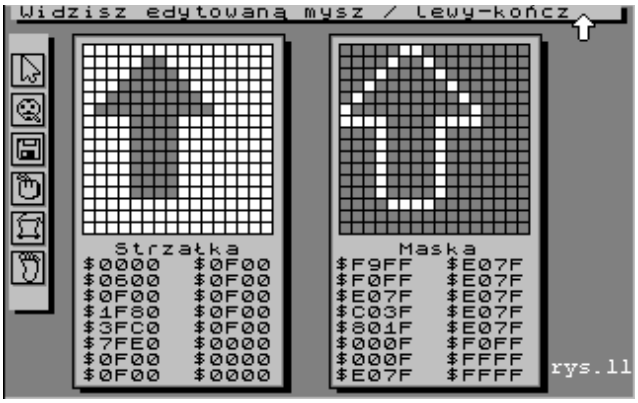
szesnastkowo **$7000**$=7\cdot16^3+0\cdot16^2+K=28672$

W zapisie definicji będziemy stosowali zapis szesnastkowy.Przenieśmy się teraz na *rys.10a* gdzie mamy definicję maski.Zerujemy wszystkie pola zajęte przez strzałkę.Aby mysz była

widoczna dobrze na dowolnym tle wyzerujemy również zarys konturu strzałki.W pozostałe pola (na rysunku najciemniejsze) wstawimy wartość 1.
Czwarta linia maski jest wobec tego opisana następująco :

binarnie **0000011111111111** ,
dziesiętnie 1·1024+1·512+1·256+1·128+1·64+1·32+1·16+1·8+1·4+1·2+1=**2047**
szesnastkowo **$07FF** .

W podobny sposób definiujemy pozostałe linii strzałki i jej maski.Uzyskane zaś liczby gromadzimy w tablicy ***type Kurs =array [1..32] of word;***



np najpierw 16 defincji dla maski a bezpośrednio potem 16 definicji dla strzałki.A oto inny,zdefiniowany kursor myszy przy pomocy programu znajdującego się na dyskietce demonstracyjnej (**ED_M_13.EXE**).Definicję tego kursora zawiera sformułowana poniżej procedura ***KSZTALT***(*jaka*:byte).Można ją odszukać pod nazwą *Prosta*.
Krótki opis działania samego programu **ED_M_13.EXE** znajduje się w dodatku III.e.

Określimy typy :

```
Type Raster_Kursora = array [0..31] of word;
               mysza=array [0..15,0..15] of byte;
```

oraz zmienne dostępne dla wszystkich procedur obsługujących mysz :

```
var pod,gad,maska,mysz :mysza;
      strzalka :Raster_Kursora;
      sx_gada,sy_gada :integer;
```

gdzie przeznaczenie tablic będzie następujące :

pod - dla przechowania tła "pod" kursorem myszy,
gad - dla przechowania definicji strzałki,
maska - dla przechowania definicji jej maski
mysz - dla spreparownego rysunku myszy,gotowego do narysowania.
strzalka- dla aktualnie obowiązującej definicji myszy (kursor+maska),

zaś zmienne **sx_gada** oraz **sy_gada** będą odpowiedzialne za położenie przemieszczanej w przyszłości myszy.Z tablicy STRZALKA można oczywiście zrezygnować,pełni ona jednak rolę pomostu między definicjami typowymi dla innych trybów a naszą obecną sytuacją.Sformułujemy najpierw procedurę przechowującą definicje kursorów myszy:

```
Procedure Ksztalt(jaka:byte);
Const
   Strzalka1:Raster_Kursora=
           ($FFFF,$9FFF,$8FFF,$87FF,$83FF,$81FF,$80FF,$887F,
            $8C3F,$801F,$81FF,$80FF,$88FF,$9CFF,$FCFF,$FFFF,
            $0000,$4000,$6000,$7000,$7800,$7C00,$6E00,$6700,
            $6380,$6FC0,$7C00,$7600,$6600,$4300,$0300,$0000);
   Paluch:Raster_Kursora=
          ($F3FF,$E1FF,$E1FF,$E1FF,$E1FF,$E00F,$E001,$8000,
           $0000,$0000,$0000,$0000,$8001,$C001,$E003,$F007,
```

```
          $0000,$0C00,$0C00,$0C00,$0C00,$0C00,$0DB0,$0DB6,
          $6FFE,$6FFE,$6FFE,$7FFE,$3FFC,$1C1C,$0FF8,$0000);
   Prosta:Raster_Kursora=
          ($F9FF,$F0FF,$E07F,$C03F,$801F,$000F,$000F,$E07F,
          $E07F,$E07F,$E07F,$E07F,$E07F,$F0FF,$FFFF,$FFFF,
          $0000,$0600,$0F00,$1F80,$3FC0,$7FE0,$0F00,$0F00,
          $0F00,$0F00,$0F00,$0F00,$0F00,$0000,$0000,$0000);
begin
  Case jaka of
    1 : strzalka:=strzalka1;
    2 : strzalka:=paluch;
    3 : strzalka:=prosta;
  end;
end;
```

Następna procedura przygotuje oddzielne tablice dla strzałki (**GAD**) i jej maski (**MASKA**) na podstawie definicji zawartej w tablicy **STRZALKA**,w których zawarte będą informacje o kolorze każdego z 256 punktów obszaru 16x16 przeznaczonego dla myszy :

```
Procedure Zrob_Kursor(kol,jaka:byte);
Var i,j,nr : integer;
    w : word;
    b : byte;
Begin
  Ksztalt(jaka);
  FillChar(gad,256,0);  FillChar(maska,256,0);
  for i:=0 to 15 do     { przygotuj oddzielną tablicę MASKA dla maski }
     begin
       w:=Strzalka[i];
       for j:=0 to 15 do
         begin
           b:=w mod 2;
           w:=w div 2;
           if b=1 then maska[i,15-j]:=255;
         end
     end;
  for i:=16 to 31 do  { przygotuj oddzielną tablicę GAD dla strzałki }
     begin
       w:=Strzalka[i];
       for j:=0 to 15 do
         begin
           b:=w mod 2;
           w:=w div 2;
           if b=1 then gad[i-16,15-j]:=kol;
         end
     end;
     Granice_M(0,319,0,199);
    { Ustal granice działania myszy  dla ekranu 320x200}
     sx_gada:=0;sy_gada:=0;
     { Mysz pokaże się w punkcie (0,0) ekranu }
End;
```

Następnym krokiem będzie procedura rysująca kursor wg podanego wyżej przepisu,zaczynając od punktu (*x,y*):

```
Procedure Kursor(x,y:integer);
var i,j,k,l:integer;
begin
   {Przygotuj kursor myszy wg przepisu}
  for i:=0 to 15 do
    for j:=0 to 15 do
       mysz[i,j]:=gad[i,j] xor (maska[i,j] and scr[y+i,x+j]);
```

```
  if y+16>199 then k:=199-y else k:=15;
              { ustal widoczną na ekranie liczbę linii }
  if x+16>320 then l:=320-x else l:=16;
              { ustal długość linii do narysowania }
  for i:=0 to k do Move(scr[i+y,x],pod[i,0],l);
              {Zapamietaj tło pod myszą}
  for i:=0 to k do Move(mysz[i,0],scr[i+y,x],l);
              {Narysuj kursor myszy}
end;
```

Kolejne procedury będą pozwalały animować narysowany kursor myszy:odtwarzanie tła pod kursorem myszy,ukrywanie i pokazywanie kursora jak również przesuwanie kursora do wskazanego punktu :

```
Procedure Pod_G(x,y:integer);   { Odtwórz tło pod myszą }
var i,k,l:integer;
begin
  if y+16>199 then k:=199-y else k:=15;
  if x+16>320 then l:=320-x else l:=16;
  for i:=0 to k do Move(pod[i,0],scr[i+y,x],l);
end;

Procedure Ukryj_G;
begin
  Pod_G(sx_gada,sy_gada);
end;

Procedure Pokaz_G;
begin
  Ustaw_M(sx_gada,sy_gada);
  Kursor(sx_gada,sy_gada);
end;

Procedure Ustaw_G(a,b :integer);
begin
  sx_gada:=a;
  sy_gada:=b;
end;
```

I na koniec procedura odpowiedzialna za animację myszy i jednocześnie zwracająca informację o aktualnym położeniu myszy i numerze wciśniętego przycisku (1 - lewy,2 - prawy,3 - środkowy,o ile jest,wpp jednocześnie lewy i prawy).

```
Procedure Ruszaj_G(var x_gada,y_gada,guzik:integer);
begin
  Pozycja_M(x_gada,y_gada,guzik);
  if ((sx_gada<>x_gada) or (sy_gada<>y_gada)) then
    begin
      Synchronizacja;
      Pod_G(sx_gada,sy_gada);
      Kursor(x_gada,y_gada);
      sx_gada:=x_gada;
      sy_gada:=y_gada;
    end;
end;
```

Załóżmy,że wszystkie omówione procedury są zgromadzone w module **MYSZGR.TPU**, którego początek jest następujący:

```
Unit MyszGr;
INTERFACE
uses crt,dos,graf_13;
Function Mysz_O:boolean;
Function WPrzycisk(nr:integer):boolean;
Procedure Pozycja_M(var x,y,przyciski:integer);
Procedure Granice_M(x_min,x_max,y_min,y_max:integer);
Procedure Ustaw_M(x,y:integer);
Procedure Ksztalt(jaka:byte);
Procedure Zrob_Kursor(kol,jaka:byte);
Procedure Kursor(x,y:integer);
Procedure Pod_G(x,y:integer);
Procedure Ruszaj_G(var x_gada,y_gada,guzik:integer);
Procedure Ukryj_G;
Procedure Pokaz_G;
Procedure Ustaw_G(a,b :integer);
IMPLEMENTATION
Type Raster_Kursora = array [0..31] of word;
                           mysza=array [0..15,0..15] of byte;
var pod,gad,maska,mysz :mysza;
      strzalka :Raster_Kursora;
      sx_gada,sy_gada :integer;
```

...

A teraz krótki program demonstrujący poprawność naszego rozwiązania :

```
Program Jest_Mysz;
uses crt,graf_13,myszgr;
var xm,ym,prz,rodz:integer;
BEGIN
  Grafika;Tlo(9);
  Linia(0,0,319,199,66);
  rodz:=1;
  Zrob_Kursor(12,rodz);
  Ustaw_G(160,100);
  Pokaz_G;
  Repeat
    Ruszaj_G(xm,ym,prz);
    if prz=1 then
      begin
        Inc(rodz);
        if rodz>3 then rodz:=1;
        Ukryj_G;
        Zrob_Kursor(11+rodz,rodz);
        Repeat Until not WPrzycisk(1);
        Ustaw_G(xm,ym);
        Pokaz_G;
      end;
  Until prz=2;
  TextMode(3);
END.
```

Program pozwala jedynie obserwować mysz oraz lewym przyciskiem zmieniać jej kształt, działanie kończymy prawym przyciskiem.Należy jeszcze nadmienić,że powyższe rozwiązanie daje przewidywany efekt animacji myszy jeśli procedurę *Ruszaj_G(xm,ym,prz)* wywołujemy w dowolnej pętli.Dla większości przypadków to rozwiązanie wystarczy,zwykle dla akcji w której mysz może być sensownie wykorzystywana możemy zorganizować iterację z warunkiem kończącym.
Poza tym inwencji Czytelników pozostawiamy ewentualne inne sposoby organizacji współpracy myszy ze swoimi programami.

2.l. Co dalej ?

Omówione w rozdziale **II** procedury mogą nie wyczerpywać (i na pewno nie wyczerpują) potrzeb w sytuacjach graficznych na jakie możemy się natknąć realizując jakieś zadanie o tej tematyce.Dla przykładu jedną z technik stosowaną w animacji jest zmiana trybu rysowania -w module ***GRAPH.TPU*** zapewnia ją procedura *SetWriteMode*(tryb).W jednym z rozdziałów pokazaliśmy konstrukcję i zasady używania dodatkowej strony pomijając milczeniem fakt,jak zapewnić realizację innych,uprzednio z dużym mozołem formułowanych procedur,tak by działały bez zarzutu na obu (lub wielu stronach).W takiej sytuacji pewnie należałoby budować unit ***GRAF_13*** (patrz dodatek **III)** zapewniając większą elastyczność wszystkim rozwiązaniom.Należałoby zacząć na przykład od tego by jednak punkt na ekranie rysować poprzez wpisywanie do adresu karty wartości koloru.
Zmienna *SCR :EKRAN absolute $A000:0* ,co prawda wygodna w zapisie procedur ,czyni jednak całe rozwiązanie dość statycznym.
Aby swobodnie korzystać ze wszystkich formułowanych procedur,na dowolnej stronie,ustalmy zmienne globalne :

SegmentE,OfsetE : word;

Teraz dla przykładu,jeśli chcemy np korzystać z dwóch stron deklarujemy dodatkową stronę :

```
VAR strona :pointer;
.....
GETMEM(strona,64000);
```

Definiujemy procedurę zmieniającą stronę :

```
PROCEDURE A_STRONA(nr:byte);
begin
  Case nr of
     0 :begin   SegmentE:=$A000;        OfsetE:=0;              end;
     1 : begin  SegmentE:=Seg(strona^);OfsetE:=Ofs(strona^);end;
end;
```

I dla kompletu rozwiązania procedurę rysującą punkt (bez sprawdzania czy mieści się na ekranie :

```
PROCEDURE Scr(x,y:integer;kolor:byte);
begin
  MEM[SegmentE:OfsetE+320*y+x]:=kolor;
end;
```

Dalej aby cały moduł poprawnie funkcjonował potrzebne są już tylko niewielkie poprawki edytorskie.
W jednym z podrozdziałów omawialiśmy funkcje BIOSU odpowiedzialne za przewijanie pewnej prostokątnej części ekranu.Dla prostokąta zawierającego elementy tekstu takie rozwiązanie może oczywiście wystarczyć,jednak przesuwanie okna z grafiką należałoby zorganizować bardziej elegancko tzn linia po linii.Problemy oczywiście możnaby jeszcze długo wymieniać.Jednak należałoby zadać sobie pytanie w jakim celu gromadzimy procedury w jednym module.Czy zawsze,w każdym programie wszystkie procedury będą wykorzystywane ?
Jeśli nie, to może na tym poprzestać zaś dla każdego nowego zadania tworzyć dodatkowe i to w ilości absolutnie niezbędnej by rozwiązać doraźnie problem nad którym aktualnie pracuję.
Odpowiedzi na te pytania oczywiście należy roztrzygnąć indywidualnie.
Jest jednak jedna rzecz,którą można i powinno się zrobić.Znając zasady opisane w tej pracy trzeba po prostu optymalizować niektóre rozwiązania i dostosowywać do swoich potrzeb.

Dla przykładu procedury obsługujące mysz o wiele lepiej będą się prezentować jeśli napiszemy je w *assemblerze* .
Należałoby wreszcie zająć się problemem wypełniania obszaru: ograniczonego dowolnym wielokątem i dowolną krzywą zamkniętą.Takie procedury często przydają się,zwłaszcza że rysunek w trybie 320x200 powinien być specyficznie konstruowany.Punkty w tym trybie są niestety dość duże i ze względów estetycznych powinniśmy operować właśnie większymi obszarami zamalowanymi lub zacieniowanymi. Zdarzają się oczywiście wyjatki.
W rozdziale pierwszym pokazujemy pakiet procedur,który organizuje ekran ze skalą rzeczywistą.Można więc zastanowić się czy teraz,w trybie 13h wprowadzać takie rozwiązanie i w jakim zakresie,przy jakich okazjach.

III.Dodatki

III.a. Wybrane funkcje przerwania 10h

Dokładny opis wszystkich funkcji przerwania 10h znajdzie Czytelnik w [1].Poniżej podajemy jedynie wykaz tych,które zostały użyte w prezentowanym opracowaniu.Zasada użycia w programie pascalowym jest następująca:

```
Procedure Korzystam_z_przerwania_10h;
var rej:Registers;
begin
  rej.AH:=nr_funkcji_przerwania;
    {innym rejestrom przypisać wymagane wartości}
  INTR($10,rej);
end;
```

Typ *Registers* jest zdefiniowany w module DOS.

Funkcja 00h - zmiana trybu ekranu

We: AH=$00
AL=numer trybu
Wy: -

Funkcja 02h - zmiana położenia kursora

We: AH=$02
BH=numer strony
DH=numer wiersza
DL=numer kolumny
Wy: -

Funkcja 06h - przewijanie okna w górę

We: AH=$06
AL=liczba wierszy do przewinięcia (jeśli 0 to wyczyść obszar)
BH=atrybut dla pustych wierszy
CH=numer górnego wiersza okna
CL=numer górnej,lewej kolumny okna
DH=numer dolnego wiersza okna
DL=numer prawej,dolnej kolumny okna
Wy: -

Funkcja 07h - przewijanie okna w dół

We: AH=$07
AL=liczba wierszy do przewinięcia (jeśli 0 to wyczyść obszar)
BH=atrybut dla pustych wierszy
CH=numer górnego wiersza okna
CL=numer górnej,lewej kolumny okna
DH=numer dolnego wiersza okna
DL=numer prawej,dolnej kolumny okna
Wy: -

Funkcja 0Ah - zapis znaku w miejscu wskazanym przez kursor

We: AH=$0A
AL=kod znaku
BH=kolor tła
BL=kolor znaku
CX=ilość powtórzeń
Wy: -

Funkcja 0Ch - rysowanie piksla
We: AH=$0C
AL=numer koloru(+$80=XOR)
BH=numer strony
CX=współrzędna X
DX=współrzędna Y
Wy: -

Funkcja ODh - odczyt koloru piksla
We: AH=$0D
BH=numer strony
CX=współrzędna X
DX=współrzędna Y
Wy: - numer koloru piksla

Funkcja 10h,podfunkcja 10h - zapis do jednego rejestru DAC
We: AH=$0010
AL=$0010
BX=kolor (numer rejestru DAC)
DH=składowa czerwona
CH=składowa zielona
CL=składowa niebieska
Wy -

Funkcja 10h,podfunkcja 15h - odczyt jednego rejestru DAC
We: AH=$0010
AL=$0015
BX=kolor (numer rejestru DAC)
Wy DH=składowa czerwona
CH=składowa zielona
CL=składowa niebieska

Funkcja 11h,podfunkcja 21h - wybór adresu definicji znaków graficznych
We: AH=$0011
AL=$0021
S:BP=adres tablicy definicji znaków 0÷255
CX=wysokość znaku (liczba bajtów na definicję)
BL=liczba wierszy na ekranie
0:DL zawiera liczbę wierszy
1:14 wierszy
2:25 wierszy
3:43 wiersze
Wy -

III.b. Tabela wartości R,G,B 256 kolorów .

kol	R	G	B	kol	R	G	B	kol	R	G	B	kol	R	G	B	kol	R	G	B
0	0	0	0	51	0	63	47	102	45	54	63	153	22	20	28	204	16	8	16
1	0	0	42	52	0	63	63	103	45	49	63	154	24	20	28	205	16	8	14
2	0	42	0	53	0	47	63	104	0	0	28	155	26	20	28	206	16	8	12
3	0	42	42	54	0	31	63	105	7	0	28	156	28	20	28	207	16	8	10
4	42	0	0	55	0	16	63	106	14	0	28	157	28	20	26	208	16	8	8
5	42	0	42	56	31	31	63	107	21	0	28	158	28	20	24	209	16	10	8
6	42	21	0	57	39	31	63	108	28	0	28	159	28	20	22	210	16	12	8
7	42	42	42	58	47	31	63	109	28	0	21	160	28	20	20	211	16	14	8
8	21	21	21	59	55	31	63	110	28	0	14	161	28	22	20	212	16	16	8
9	21	21	63	60	63	31	63	111	28	0	7	162	28	24	20	213	14	16	8
10	21	63	21	61	63	31	55	112	28	0	0	163	28	26	20	214	12	16	8
11	21	63	63	62	63	31	47	113	28	7	0	164	28	28	20	215	10	16	8
12	63	21	21	63	63	31	39	114	28	14	0	165	26	28	20	216	8	16	8
13	63	21	63	64	63	31	31	115	28	21	0	166	24	28	20	217	8	16	10
14	63	63	21	65	63	39	31	116	28	28	0	167	22	28	20	218	8	16	12
15	63	63	63	66	63	47	31	117	21	28	0	168	20	28	20	219	8	16	14
16	0	0	0	67	63	55	31	118	14	28	0	169	20	28	22	220	8	16	16
17	5	5	5	68	63	63	31	119	7	28	0	170	20	28	24	221	8	14	16
18	8	8	8	69	55	63	31	120	0	28	0	171	20	28	26	222	8	12	16
19	11	11	11	70	47	63	31	121	0	28	7	172	20	28	28	223	8	10	16
20	14	14	14	71	39	63	31	122	0	28	14	173	20	26	28	224	11	11	16
21	17	17	17	72	31	63	31	123	0	28	21	174	20	24	28	225	12	11	16
22	20	20	20	73	31	63	39	124	0	28	28	175	20	22	28	226	13	11	16
23	24	24	24	74	31	63	47	125	0	21	28	176	0	0	16	227	15	11	16
24	28	28	28	75	31	63	55	126	0	14	28	177	4	0	16	228	16	11	16
25	32	32	32	76	31	63	63	127	0	7	28	178	8	0	16	229	16	11	15
26	36	36	36	77	31	55	63	128	14	14	28	179	12	0	16	230	16	11	13
27	40	40	40	78	31	47	63	129	17	14	28	180	16	0	16	231	16	11	12
28	45	45	45	79	31	39	63	130	21	14	28	181	16	0	12	232	16	11	11
29	50	50	50	80	45	45	63	131	24	14	28	182	16	0	8	233	16	12	11
30	56	56	56	81	49	45	63	132	28	14	28	183	16	0	4	234	16	13	11
31	63	63	63	82	54	45	63	133	28	14	24	184	16	0	0	235	16	15	11
32	0	0	63	83	58	45	63	134	28	14	21	185	16	4	0	236	16	16	11
33	16	0	63	84	63	45	63	135	28	14	17	186	16	8	0	237	15	16	11
34	31	0	63	85	63	45	58	136	28	14	14	187	16	12	0	238	13	16	11
35	47	0	63	86	63	45	54	137	28	17	14	188	16	16	0	239	12	16	11
36	63	0	63	87	63	45	49	138	28	21	14	189	12	16	0	240	11	16	11
37	63	0	47	88	63	45	45	139	28	24	14	190	8	16	0	241	11	16	12
38	63	0	31	89	63	49	45	140	28	28	14	191	4	16	0	242	11	16	13
39	63	0	16	90	63	54	45	141	24	28	14	192	0	16	0	243	11	16	15
40	63	0	0	91	63	58	45	142	21	28	14	193	0	16	4	244	11	16	16
41	63	16	0	92	63	63	45	143	17	28	14	194	0	16	8	245	11	15	16
42	63	31	0	93	58	63	45	144	14	28	14	195	0	16	12	246	11	13	16
43	63	47	0	94	54	63	45	145	14	28	17	196	0	16	16	247	11	12	16
44	63	63	0	95	49	63	45	146	14	28	21	197	0	12	16	248	0	0	0
45	47	63	0	96	45	63	45	147	14	28	24	198	0	8	16	249	0	0	0
46	31	63	0	97	45	63	49	148	14	28	28	199	0	4	16	250	0	0	0
47	16	63	0	98	45	63	54	149	14	24	28	200	8	8	16	251	0	0	0
48	0	63	0	99	45	63	58	150	14	21	28	201	10	8	16	252	0	0	0
49	0	63	16	100	45	63	63	151	14	17	28	202	12	8	16	253	0	0	0
50	0	63	31	101	45	58	63	152	20	20	28	203	14	8	16	254	0	0	0

III.f. Listing moduł GRAF 13 dla grafiki trybu 13h.

```
Unit Graf_13;

INTERFACE
Uses Crt,Dos;
{=========================================================}
const skala=0.85;
      x_aspekt:longint=22;
      y_aspekt:longint=25;
Type Ekran= array [0..199,0..319] of byte;
var scr : Ekran absolute $A000:0000;
    tlo_:byte;
{---------------------------------------------------------}
Procedure AT(kol,wier:byte);
Procedure Bar_(x1,y1,x2,y2:integer;kolor:word);
Function Brzeg(p,q :integer;var t1,t2:real):boolean;
Procedure Cien(x,y,dl,sz:integer;kolor:byte);
Procedure Circle(xo,yo,r:integer;k :word);
Procedure Czekaj;
Procedure Elipsa(a,b,os1,os2:integer;kolor:byte);
Procedure Grafika;
Procedure Jakie_RGB(kolor:byte;var r,g,b:byte);
Procedure KoniecGrafiki;
Procedure Kreska(xp,yp,xk,yk,k :integer);
Procedure Linia(xp,yp,xk,yk,kol:integer);
Procedure Okrag(a,b,r:integer;kol:byte);
Procedure Pas(wier,kol:byte);
Procedure Piksel(x,y:integer;kolor:word);
Procedure Punkt(x,y:integer;kolor:word);
Procedure Pisz(asc:byte;kol_z,kol_t:word);
Procedure Przypomnij(a,b,dl,sz : integer; pr : pointer);
Procedure PrzypomnijEkran(pr:pointer);
Procedure RGB(kolor,r,g,b:byte);
Procedure Synchronizacja;
Procedure WgrajEkran(nazwa:string);
Procedure Wpisz(kol,wier,kol_napisu,kol_tla:byte;zd:string);
Procedure Tlo(kolor:word);
Procedure Zwolnij(a,b,dl,sz : integer; pr : pointer);
Procedure Zapamietaj(a,b,dl,sz : integer; var pr : pointer);
Procedure ZapamietajEkran(var pr:pointer);
Procedure ZwolnijEkran(pr:pointer);
Procedure ZapiszEkran(nazwa:string);
{---------------------------------------------------------}
IMPLEMENTATION
{---------------------------------------------------------}
Procedure AT(kol,wier:byte);
var Rej :Registers;
begin
  With Rej do
    begin
      AH:=$02;
      BH:=0;
      DH:=wier;
      DL:=kol;
    end;
    INTR($10,Rej);
end;
{---------------------------------------------------------}
Procedure Bar_(x1,y1,x2,y2:integer;kolor:word);
var i,dl,il:integer;
begin
   if ((x2<x1) or (y2<y1)) then Exit;
   if x1<0 then x1:=0;
   if x1>319 then x1:=319;
   if y1<0 then y1:=0;
   if y1>199 then y1:=199;
   dl:=x2-x1;
   if x1+dl>319 then dl:=319-x1;
   il:=y2-y1;
   if y1+il>199 then il:=199-y1;
   FillChar(scr[y1,x1,dl,kolor);
   for i:=y1+1 to y1+il-1 do
      begin
        Move(scr[y1,x1],scr[i,x1],dl);
      end;
end;
```

```
    {------------Test-Lianga_Barsky'ego---------------}
{-------------------------------------------------------}
Function Brzeg(p,q :integer;var t1,t2:real):boolean;
var fakt:boolean;
    t:real;
begin
  fakt:=true;
  if p<0 then
    begin
       t:=q/p;
       if t>t2 then fakt:=false
         else
          if t>t1 then t1:=t;
    end
    else
  if p>0 then
    begin
       t:=q/p;
       if t<t1 then fakt:=false
         else
          if t<t2 then t2:=t;
    end
     else
  if q<0 then fakt:=false;
  Brzeg:=fakt;
end;
{-------------------------------------------------------}
Procedure Cien(x,y,dl,sz:integer;kolor:byte);
var i,j:integer;
begin
  for i:=1 to sz do
    for j:=1 to dl do
        scr[y+i-1,x+j-1]:=((scr[y+i-1,x+j-1] xor tlo_) xor kolor);
end;
{---------Okrąg-wg-algorytmu-Bresenhama-bez_aspektu--}
{-------------------------------------------------------}
Procedure Circle(xo,yo,r:integer;k:word);
var p,x,y :integer;
     Procedure RysPkt(xo,yo,x,r:integer;k:word);
     begin
         Piksel(xo+x,yo+r,k);Piksel(xo-x,yo+r,k);
         Piksel(xo+x,yo-r,k);Piksel(xo-x,yo-r,k);
         Piksel(xo+r,yo+x,k);Piksel(xo-r,yo+x,k);
         Piksel(xo+r,yo-x,k);Piksel(xo-r,yo-x,k);
     end;
begin
  x:=0;y:=r;p:=3-2*r;
  Repeat
    RysPkt(xo,yo,x,y,k);
    if p<0 then p:=p+4*x+6
      else
        begin
          p:=p+4*(x-y)+10;
          Dec(y);
        end;
        Inc(x);
  Until x>y;
end;
{-------------------------------------------------------}
Procedure Czekaj;
var ch :char;
begin
  Repeat
    ch:=ReadKey;
  Until ch=#27;
end;
{-------------------------------------------------------}
Procedure Elipsa(a,b,os1,os2:integer;kolor:byte);
begin
  y_aspekt:=Round(25*os1/os2);
  Okrag(a,b,os2,kolor);
  y_aspekt:=25;
end;
{-------------------------------------------------------}
Procedure Grafika;
var rej :Registers;
begin
```

```
  rej.AX:=$0013;
  Intr($10,rej);
  tlo_:=0;
end;
{-------------------------------------------------------}
Procedure Jakie_RGB(kolor:byte;var r,g,b:byte);
begin
  Port[$3C7]:=kolor;
  r:=Port[$3C9];
  g:=Port[$3C9];
  b:=Port[$3C9];
end;
{-------------------------------------------------------}
Procedure Kreska(xp,yp,xk,yk,k :integer);
var dx,dy,d,x,y,d1,d2,sx,sy :integer;
begin
  dx:=abs(xk-xp);
  dy:=abs(yk-yp);
  if xk>=xp then sx:=1 else sx:=-1;
  if yk>=yp then sy:=1 else sy:=-1;
  x:=xp;
  y:=yp;
  if dx>=dy then
     begin
       d:=2*dy-dx;
       d1:=2*dy;
       d2:=2*(dy-dx);
       scr[y,x]:=k;
       while (x<>xk) do
          begin
            if d<0 then
              begin
                Inc(d,d1);
                Inc(x,sx);
              end
                else
                   begin
                     Inc(x,sx);
                     Inc(y,sy);
                     Inc(d,d2);
                   end;
             scr[y,x]:=k;
           end
      end
        else
          begin  {dx<dy}
            d:=2*dx-dy;
            d1:=2*dx;
            d2:=2*(dx-dy);
            scr[y,x]:=k;
            while (y<>yk) do
              begin
                if d<0 then
                  begin
                    Inc(d,d1);
                    Inc(y,sy);
                  end
                    else
                      begin
                        Inc(x,sx);
                        Inc(y,sy);
                        Inc(d,d2);
                      end;
                scr[y,x]:=k;
              end
          end;
end;
{-------------------------------------------------------}
Procedure KoniecGrafiki;
begin
  TextMode(3);
end;
{-------------------------------------------------------}
Procedure Linia(xp,yp,xk,yk,kol:integer);
var t1,t2:real;
    px,py,x,y:integer;
begin
```

```
  px:=xk-xp;
  py:=yk-yp;
  x:=xp;y:=yp;
  t1:=0;t2:=1;
  if Brzeg(-px,xp,t1,t2) then
    if Brzeg(px,319-xp,t1,t2) then
      if Brzeg(-py,yp,t1,t2) then
        if Brzeg(py,199-yp,t1,t2) then
        begin
          if t1>0 then
            begin
              xp:=Round(xp+t1*px);
              yp:=Round(yp+t1*py);
            end;
          if t2<1 then
            begin
              xk:=Round(x+t2*px);
              yk:=Round(y+t2*py);
            end;
          Kreska(xp,yp,xk,yk,kol);
      end;
end;
   {--------Okrąg-wg-algorytmu-Bresenhama-z_aspektem }
{-------------------------------------------------------}
Procedure Okrag(a,b,r:integer;kol:byte);
var pp,qq,pp4,qq4,pp8,qq8,fx,fy,fs,x,y:longint;

   Procedure RysPkt(xo,yo,x,y:integer;k:word);
     begin
         Punkt(xo+x,yo+y,k);Punkt(xo-x,yo+y,k);
         Punkt(xo+x,yo-y,k);Punkt(xo-x,yo-y,k);
     end;

begin
  x:=0;y:=r;
  pp:=sqr(x_aspekt);
  qq:=sqr(y_aspekt);
  pp4:=4*pp;
  qq4:=4*qq;
  pp8:=8*pp;
  qq8:=8*qq;
  fx:=0;
  fy:=qq8*r;fs:=pp4-qq4*r+qq;
  While fx<fy do
    begin
      RysPkt(a,b,x,y,kol);
      Inc(x);
      fx:=fx+pp8;
      if fs<=0 then fs:=fs+fx+pp4
        else
          begin
            Dec(y);
            fy:=fy-qq8;
            fs:=fs+fx+pp4-fy;
          end;
    end;
    fs:=fs-((fx+fy) div 2)+3*(pp-qq);
    While y>=0 do
      begin
        RysPkt(a,b,x,y,kol);
        Dec(y);
        fy:=fy-qq8;
        if fs<=0 then
          begin
            Inc(x);
            fx:=fx+pp8;
            fs:=fs+fx-fy+qq4;
          end
           else fs:=fs-fy+qq4;
      end;
end;
{-------------------------------------------------------}
Procedure Pas(wier,kol:byte);
var i:integer;
begin
  for i:=0 to 39 do Wpisz(i,wier,kol,kol,' ');
end;
```

```
{-------------------------------------------------------}
Procedure Piksel(x,y:integer;kolor:word);
begin
  if ((x>0) and (x<320) and (y>=0) and (y<235)) then
     scr[Round(skala*y),x]:=kolor;
end;
{-------------------------------------------------------}
Procedure Punkt(x,y:integer;kolor:word);
begin
  if ((x>0) and (x<320) and (y>=0) and (y<200)) then
     scr[y,x]:=kolor;
end;
{-------------------------------------------------------}
Procedure Pisz(asc:byte;kol_z,kol_t:word);
var rej:Registers;
begin
  With Rej do
    begin
      AH:=$0A;
      AL:=asc;
      BH:=kol_t;    { kolor tła }
      BL:=kol_z;
      CX:=1;        { ilość powtórzeń }
    end;
   INTR($10,Rej);
end;
{-------------------------------------------------------}
Procedure PrzypomnijEkran(pr:pointer);
begin
  Move(pr^,scr,64000);
end;
{-------------------------------------------------------}
Procedure RGB(kolor,r,g,b:byte);
begin
  Port[$3C8]:=kolor;
  Port[$3C9]:=r;
  Port[$3C9]:=g;
  Port[$3C9]:=b;
end;
{-------------------------------------------------------}
Procedure Synchronizacja;
begin
   While (Port[$3DA] and 8)=0 do;
end;
{-------------------------------------------------------}
Procedure WgrajEkran(nazwa:string);
var plik:file of Ekran;
begin
  Assign(plik,nazwa);
  Reset(plik);
  read(plik,scr);
  Close(plik);
end;
{-------------------------------------------------------}
Procedure Wpisz(kol,wier,kol_napisu,kol_tla:byte;zd:string);
var i:byte;
begin
  for i:=1 to length(zd) do
    begin
      AT(kol+i-1,wier);
      Pisz(Ord(zd[i]),kol_napisu,kol_tla);
    end;
end;
{-------------------------------------------------------}

Procedure Tlo(kolor:word);
begin
  FillChar(scr,200*320,kolor);
  tlo_:=kolor;
end;
{-------------------------------------------------------}
Procedure Zapamietaj(a,b,dl,sz : integer; var pr : pointer);
var i:integer;
begin
   dl:=dl+1;
   if a+dl>319 then dl:=319-a;
   if b+sz>199 then sz:=199-b;
```

```
  GetMem(pr,dl*sz);
  for i:=0 to sz-1 do
      Move(scr[b+i,a],PTR(Seg(pr^),Ofs(pr^)+dl*i)^,dl);
end;
{-------------------------------------------------------}
Procedure Przypomnij(a,b,dl,sz : integer; pr : pointer);
var i : integer;
begin
   dl:=dl+1;
   if a+dl>319 then dl:=319-a;
   if b+sz>199 then sz:=199-b;
   for i:=0 to sz-1 do
      Move(PTR(Seg(pr^),Ofs(pr^)+dl*i)^,scr[b+i,a],dl);
end;
{-------------------------------------------------------}
Procedure Zwolnij(a,b,dl,sz : integer; pr : pointer);
begin
   dl:=dl+1;
   sz:=sz;
   if a+dl>319 then dl:=319-a;
   if b+sz>199 then sz:=199-b;
  FreeMem(pr,dl*sz)
end;
{-------------------------------------------------------}
Procedure ZapamietajEkran(var pr:pointer);
begin
  GetMem(pr,64000);
  Move(scr,pr^,64000)
end;
{-------------------------------------------------------}
Procedure ZwolnijEkran(pr:pointer);
begin
  FreeMem(pr,64000);
end;
{-------------------------------------------------------}
Procedure ZapiszEkran(nazwa:string);
var plik:file of Ekran;
begin
  Assign(plik,nazwa);
  Rewrite(plik);
  write(plik,scr);
  Close(plik);
end;
{-------------------------------------------------------}
BEGIN
  Grafika;
END.
```

III.g. Listing moduł MYSZGR dla obsługi myszy w trybie 13h

```
Unit MyszGr;
INTERFACE
uses dos,graf_13;
Type Raster_Kursora = array [0..31] of word;
var sx_gada,sy_gada :integer;
    strzalka :Raster_Kursora;
Function Mysz_O:boolean;
Function WPrzycisk(nr:integer):boolean;
Procedure Pozycja_M(var x,y,przyciski:integer);
Procedure Granice_M(x_min,x_max,y_min,y_max:integer);
Procedure Ustaw_M(x,y:integer);
Procedure Ksztalt(jaka:byte);
Procedure Zrob_Kursor(kol,jaka:byte);
Procedure Kursor(x,y:integer);
Procedure Pod_G(x,y:integer);
Procedure Ruszaj_G(var x_gada,y_gada,guzik:integer);
Procedure Ukryj_G;
Procedure Pokaz_G;
Procedure Ustaw_G(a,b :integer);
{-------------------------------------------------------}
IMPLEMENTATION
{-------------------------------------------------------}
Type mysza=array [0..15,0..15] of byte;
     var pod,gad,maska,mysz :mysza;
{-------------------------------------------------------}
```

```
Function Mysz_O:boolean;
var rej:Registers;
begin
  rej.AX:=0;
  Intr($33,rej);
  if rej.AX=0 then Mysz_O:=false
    else Mysz_O:=true;
end;
{-------------------------------------------------------}
Function WPrzycisk(nr:integer):boolean;
var rej :Registers;
begin
  rej.AX:=3;
  Intr($33,rej);
  if (rej.BX and (1 shl(nr-1)))=0
    then WPrzycisk:=false
      else WPrzycisk:=true;
end;
{-------------------------------------------------------}
Procedure Pozycja_M(var x,y,przyciski:integer);
var rej:Registers;
begin
  rej.AX:=3;
  Intr($33,rej);
  x:=rej.CX;
  y:=rej.DX;
  przyciski:=rej.BX;
end;
{-------------------------------------------------------}
Procedure Granice_M(x_min,x_max,y_min,y_max:integer);
var rej:Registers;
begin
  rej.AX:=7;
  rej.CX:=x_min;
  rej.DX:=x_max;
  Intr($33,rej);

  rej.AX:=8;
  rej.CX:=y_min;
  rej.DX:=y_max;
  Intr($33,rej);
end;
{-------------------------------------------------------}
Procedure Ustaw_M(x,y:integer);
var rej:Registers;
begin
  rej.AX:=4;
  rej.CX:=x;
  rej.DX:=y;
  Intr($33,rej);
end;
{-------------------------------------------------------}
Procedure Ksztalt(jaka:byte);
Const
   Strzalka1:Raster_Kursora=
           ($FFFF,$9FFF,$8FFF,$87FF,$83FF,$81FF,$80FF,$887F,
            $8C3F,$801F,$81FF,$80FF,$88FF,$9CFF,$FCFF,$FFFF,
            $0000,$4000,$6000,$7000,$7800,$7C00,$6E00,$6700,
            $6380,$6FC0,$7C00,$7600,$6600,$4300,$0300,$0000);
   Paluch:Raster_Kursora=
          ($F3FF,$E1FF,$E1FF,$E1FF,$E1FF,$E00F,$E001,$8000,
           $0000,$0000,$0000,$0000,$8001,$C001,$E003,$F007,
           $0000,$0C00,$0C00,$0C00,$0C00,$0C00,$0DB0,$0DB6,
           $6FFE,$6FFE,$6FFE,$7FFE,$3FFC,$1C1C,$0FF8,$0000);
   Prosta:Raster_Kursora=
          ($F9FF,$F0FF,$E07F,$C03F,$801F,$000F,$000F,$E07F,
          $E07F,$E07F,$E07F,$E07F,$E07F,$F0FF,$FFFF,$FFFF,
          $0000,$0600,$0F00,$1F80,$3FC0,$7FE0,$0F00,$0F00,
          $0F00,$0F00,$0F00,$0F00,$0F00,$0000,$0000,$0000);
begin
  Case jaka of
    1 : strzalka:=strzalka1;
    2 : strzalka:=paluch;
    3 : strzalka:=prosta;
  end;
end;
{-------------------------------------------------------}
```

```
Procedure Zrob_Kursor(kol,jaka:byte);
Var i,j,nr : integer;
    w : word;
    b : byte;
Begin
  Ksztalt(jaka);
  FillChar(gad,256,0);
  FillChar(maska,256,0);
  for i:=0 to 15 do
     begin
       w:=Strzalka[i];
       for j:=0 to 15 do
         begin
           b:=w mod 2;
           w:=w div 2;
           if b=1 then maska[i,15-j]:=255;
         end
     end;
  for i:=16 to 31 do
     begin
       w:=Strzalka[i];
       for j:=0 to 15 do
         begin
           b:=w mod 2;
           w:=w div 2;
           if b=1 then gad[i-16,15-j]:=kol;
         end
     end;
     Granice_M(0,319,0,199);
     sx_gada:=0;sy_gada:=0;
End;
{-------------------------------------------------------}
Procedure Kursor(x,y:integer);
var i,j,k,l:integer;
begin
  for i:=0 to 15 do            {Przygotuj kursor myszy wg przepisu}
    for j:=0 to 15 do
       mysz[i,j]:=gad[i,j] xor (maska[i,j] and scr[y+i,x+j]);

  if y+16>199 then k:=199-y else k:=15;
  if x+16>320 then l:=320-x else l:=16;
  for i:=0 to k do Move(scr[i+y,x],pod[i,0],l);    {Zapamietaj tło pod myszą}
  for i:=0 to k do Move(mysz[i,0],scr[i+y,x],l);   {Narysuj kursor myszy}
end;
{-------------------------------------------------------}
Procedure Pod_G(x,y:integer);
var i,k,l:integer;
begin
  if y+16>199 then k:=199-y else k:=15;
  if x+16>320 then l:=320-x else l:=16;
  for i:=0 to k do Move(pod[i,0],scr[i+y,x],l);   {Odtwórz tło pod myszą}
end;
{-------------------------------------------------------}
Procedure Ruszaj_G(var x_gada,y_gada,guzik:integer);
begin
  Pozycja_M(x_gada,y_gada,guzik);
  if ((sx_gada<>x_gada) or (sy_gada<>y_gada)) then
    begin
      Synchronizacja;
      Pod_G(sx_gada,sy_gada);
      Kursor(x_gada,y_gada);
      sx_gada:=x_gada;sy_gada:=y_gada;
    end;
end;
{-------------------------------------------------------}
Procedure Ukryj_G;
begin
  Pod_G(sx_gada,sy_gada);
end;
{-------------------------------------------------------}
Procedure Pokaz_G;
begin
  Ustaw_M(sx_gada,sy_gada);
  Kursor(sx_gada,sy_gada);
end;
{-------------------------------------------------------}
Procedure Ustaw_G(a,b :integer);
```

```
begin
  sx_gada:=a;
  sy_gada:=b;
end;
{-------------------------------------------------------}
end.
```