unit IIUWGRAPH: class;

{ this predefined class enables basic graphic operations
for DOS machines based on **486** or 386 processors }

{ this document gives the specification of new version of IIUWGRAPH
class made in October 1994 by ***Frederic Pataud*** à Pau
}

{ the early versions of library IIUWGRAPH have been elaborated by
Piotr Carlsson, Miroslawa Milkowska, Janina Jankowska,
Michal Jankowski at Institute of Informatics,
University of Warsaw 1987,
and added to Loglan system by Danuta Szczepanska 1987,

the recent versions were done at LITA, Pau,
by
Pawel Susicki (1991) for Unix,
Sebastien Bernard (1992) for ATARI, see a separate document,
Eric Becourt et Jerôme Larrieu (1993) for Unix and Xwindows, see a
separate document on Xiiuwgraf ,

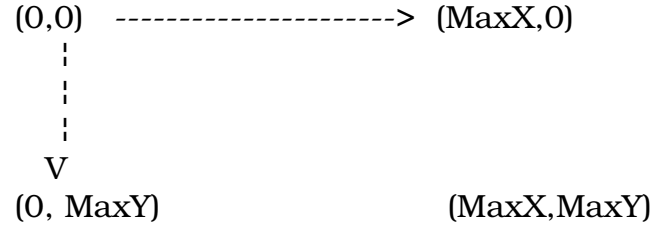
fait à Pau, le 15 Novembre 1994, par Andrzej Salwicki, LITA}

{ the predefined class IIUWGRAPH is included in all versions of interpreter of Loglan, with the *exception* of the present version of interpreter for VAX/VMS.}

hidden MaxX, MaxY, current_X, current_Y, is_graphic_On,
current_Colour, current_Background_Colour, current_Style,
current_Palette, current_Pattern ;

const MaxX =
MaxY =

{ the screen's coordinates are



}

var currentDriver : integer, { see NOCARD below }
current_X, current_Y: integer { it is the current position }
is_graphic_On: Boolean, { evidently tells whether we are in graphics mode }
current_Colour : integer, { }
current_Background_Colour : integer,
current_Style : integer, { }
current_Palette : integer,
current_Pattern

unit GRON : **procedure** (i: integer);
{ procedure sets the monitor in graphic mode and clears the buffer of screen. The parameter determines the resolution and the number of colours.
The user should assure that the resolution chosen should correspond to that which set by means of command
SET go32 drivers {path}<driver.file> <width> <height><noColours>
eg.
set go32 drivers c:\loglan\svga\drivers\vesa.grn gw 1024 gh 480 nc 256
An execution of instruction call gron(i) ***must precede*** any of the graphic commands described below.
}

unit GROFF : **procedure**;
{ the procedure sets the monitor in the text mode filling it with spaces.
DO NOT FORGET to set the monitor in the text mode before you terminate your program
}

~~unit NOCARD : function : integer;~~
~~{ the value given by this function determines the type of the currently used monitor and it is equal to~~
~~1 for Hercules mono card,~~
~~2 for IBM CGA color~~
~~3 for IBM CGA mono 320 x 200~~
~~4 for IBM CGA mono 640 x 200~~
~~5 for EGA/VGA card~~
~~6 for ATARI STE~~
~~7 for Unix versions equipped with XWindows~~
~~*You can not call the function nocard before GRON sets the graphic mode*~~
~~}~~

unit CLS : **procedure**;
{ the screen will be cleared and filled with colour 0 }

unit VIDEO : procedure(A: array of integer);
{ this procedure can not be applied with egaint = EGA/VGA card }
{ the worktime buffer will be associated with the array A.
A call of VIDEO does not change the contents of the buffer.
All subsequent calls of the procedures modifying the screen will
concern the array A. The screen does not change.
A ready image can be moved to the screen with the help of
GETMAP/PUTMAP procedures or it can be stored on disk.
The array should have 16 kBytes for IBM CGA card or
32 kBytes for Hercules card.}

{ PROCEDURES CONTROLLING THE COLOURS }

unit COLOR : **procedure**(co : integer);
{ sets current color to co
for monochrome displays, 0 means black, non-0 - white
for color displays, 0 means background
see PALLET
}

unit STYLE : **procedure**(styl : integer);
{ sets style of lines and fill shades to a combination
of current color and background color (for mono -
white and black, respectively) according to 5 predefined
patterns:

0	....
1	****
2	***.
3	**..
4	*.*.
5	*...

where '*' means current color, '.' background colour
When drawing the segments the subsequent pixels will have colour determined by cyclic application of style pattern. The first and the last pixels of a segment will have always current colour.
When filling contours the given style will be applied to horizontal lines with even coordinate. The style for odd lines is determined automatically.
The same applies for perpendicular lines.
}

unit BORDER : **procedure** (background_Colour: integer);

{ sets actual background color to i (i = 0,1,...,15) }

unit PALLET : **procedure** (nr : integer);
{

the codes of colors are as follows

0	black
1	blue dark
2	green dark
3	turquoise dark
4	red dark
5	violet
6	brown
7	grey light
8	grey dark
9	blue
10	green
11	turquoise
12	red light
13	rose
14	yellow
15	white

}

{ PROCEDURES CONTROLLING POSITION }

unit MOVE : **procedure** (x,y :integer);
{ procedure MOVE sets the current position on the screen on the pixel with coordinates
x - column,
y - line }
{ precondition of MOVE:
0≤x≤MaxX & 0≤y≤MaxY
}

unit INXPOS : **function**: integer;
{ function INXPOS returns the x coordinate of the current position }

unit INYPOS : **function** : integer;
{ function INYPOS returns the y coordinate of the current position }

unit PUSHXY : **procedure**;
{ pushes current position, color & style onto the stack.
The stack is kept internally, max depth is 16
}

unit POPXY: **procedure**;

```
{       restores position, color & style from internal stack   }

{ Example
unit DIAGONAL : procedure;
   var ix, iy : integer;
begin
        call PUSHXY;
        ix := INXPOS;
        iy := INYPOS;
        call DRAW(ix+10, iy+10);
        call POPXY
end DIAGONAL;
}



{ PROCEDURES SERVING POINTS & LINES}

unit POINT : procedure(x,y: integer);
{               moves current position to pixel (x,y) and sets it to the current color
 }

unit INPIX : function (x,y : integer) : integer;
        {
                moves to pixel (x,y) and returns its color setting;
        }


unit DRAW : procedure( x,y : integer);
        {
        draws a line from current screen position to (x,y);
        sets current position to (x,y);
        line is drawn in current color, with both terminal pixels
        always turned white ( non-background) for non-black
        ( non-background ) line color.
        Bresenham's algorithm is used, pixels belonging to the segment
        change their state depending on current colour and style.
        }

unit intens: procedure(Size :integer; xCoord,yCoord:arrayof integer,
                Colour,Filled :integer);
/* draw a polygon*/
{ draw a simple, closed polygon of Size points, the edges of the polygon go from
(xCoord[i], yCoord[i]) to (xCoord[i+1], yCoord[i+1]) for i = 1, ..., Size-1
The colour used will be Colour. The polygon will be filled iff Filled<>0.
}

unit CIRB : procedure (xi, yi, rx,ry : integer, alfa, beta : real,
                                        cbord, fill : integer);
```

{
draws a circle (or ellipse, depending on aspect value, see below),
optionally filling its interior;
does not preserve position;
(*xi,yi*) - are center coordinates,
rx - radius in pixels (horizontally),
ry - radius in pixels (perpendicularly),
alfa, beta - starting & ending angles; if alfa=beta a full
circle is drawn; values should be given in radians;
cbord - border color,
fill - if fill <>0, interior is filled in current style&color
}

unit hfill: **procedure**(x : integer);
{ draw an horizontal line between the current position and
(x,currentY) with the current color, after it change the current
position to (x, currentY)
}

unit vfill: **procedure**(y : integer);
{ draw a vertical line between the current position and
(currentX,y) with the current color, after it change the current
position to (currentX,y)
}

unit patern: **procedure**(x1,y1,x2,y2,c,b : integer);
{ draw a ***rectangle*** between the points (*x1,y1*) and (*x2,y2*) with the
color *c* (the current color is not change). if *b*=0 then the box is
empty else it is filled.
}

{ Procedures operating on bitmaps }

unit GETMAP : **function** (x,y : integer) : **arrayof** integer;
{saves rectangular area between current position as
top left corner and (ix,iy) as bottom right corner,
including border lines;
position remains unchanged.
array of integer should have
4+(rows·⌈columns/8⌉·coeff)
bytes. The coefficient coeff is 1 for Hercules, 2 for CGA, 4 for EGA
card.
ATTENTION: in DOS 286 environment a bigger size of the array may
necessitate the use of *loglan* with *the option H*+, see also memavail
}

unit PUTMAP : **procedure** (a: **arrayof** integer);
{sets rectangular area of screen pixels to that saved
by "getmap" in "iarray";
same size is restored, with top left corner in current

position;
position remains unchanged.
}

unit ORMAP : **procedure** (a : **arrayof** integer);
{same as putmap, but saved bitmap is or'ed into screen
rather than just set.
}

unit XORMAP : **procedure** (a: **arrayof** integer);
{same as putmap, but saved bitmap is xor'ed into screen
rather than just set.
}

{Procedures operating on characters and strings}

unit outstring: **procedure**(x,y: integer, s: string, back_col, front_col: integer);
{ *x, y* are the coordinates where to put the string,
s is the string to be shown, in *front_col* colour letters on the *back_col* colour background
}

unit track: **procedure**(x,y,c,valeur : integer);

{ write an integer value *valeur* at the position (*x,y*) with the color *c*.
It does not change the current position nor the current color
}

unit inkey : **function** : integer;

{ returns next character from keyboard buffer;
0 is returned if buffer is empty;
special keys are returned as negative numbers;
ALT-NUM method may be used for entering character codes
above 127 (this makes entering special keys 128-132
impossible);
if a character is returned, it is also removed
from the buffer, so MS-DOS will not see it (CTRL-C!);
typeahead is allowed, echo is suppressed.
}

unit HASCII : **procedure**(c: integer);
{'xor's the character = chr(c) in a ~~8*8~~ box with top left corner
in the current position;
moves current position by ~~(8,0)~~;
call hascii(0)- sets complete box to black (=background),
with no change in position.
~~BIOS ROM font for IBM color card is used. If the font
table is not at F000:FA6E, the character will probably~~

~~be unrecognizable, and most certainly wrong.~~
~~For codes >127, table pointed to by interrupt vector 31~~
~~is used.~~ }

unit hfont: **function**(x,y,lg,min,max,default,col_f,col_e,col_c : integer):
integer;

{arrange a small 1 line window for ***reading*** an integer value from this window,
the position of the window corner is (*x, y*),
the length of the window is *lg* characters,
the value v should be greater than *min* and smaller than *max*,
the default value read is *default*,
the colour of the window is *col_f*,
the colour of the digits is *col_e,*
the colour of cursor is *col_c*

reads in graphic mode an integer in the window which begins at the (x,y) position, window is lg caracteres long. the maximum length of the integer that is read is 10. there is a default value, a minimum value and a maximum value. the window is drawn with the col_f color, the cursor is in the col_c color and the integer is writing in the col_e color. you can use 0..9,+,-,backspace,escape and return keys. }

unit HPAGE : **procedure**(x,y,long: integer, A: arrayof char, back, front: integer);
{ this procedure arranges a 1-line high window in position *x,y* of length *long* in which a portion of text *A* is shown in colour *front* on the background colour *back*.
Making use of keys controlling the cursor {left, right, PgUp, PgDn} the user can scroll the text (horizontally) in the window. Pressing the Enter key terminates the procedure}

end IIUWGRAPH;

unit MOUSE: **class**;

{ *init* -lors de l'initialisation de la souris, on peut définir les événements qui vont faire réagir la fonction getpress; le premier et le deuxième paramètre représentent respectivement la souris et le clavier, si une valeur non nulle est donnée comme paramètre alors getpress réagira à l'événement.

Une paire (1,1) va permettre de prendre en compte à la fois les événements de la souris et ceux du clavier; une paire (1,0) quand à elle ne prendra en compte que la souris. Pour une plus grande souplesse d'utilisation, il est possible lors du programme, après l'initalisation, de changer cette prise en compte, cela se fera par l'appel de la procedure *getmovement*, procédure ayant les mêmes paramètres (avec le même ordre) que la fonction init.

Pour detecter les événements, on utilisa la fonction *getpress*, qui retourne un booléen indiquant la présence ou l'absence d'événement (respectivement les valeurs true et false). Il est bon de noter qu'ainsi définie la fonction getpress n'est pas bloquante. Les paramètres en retour sont soit nuls (pas d'événement) soit correspondent:

bool:=getpress(v,p,h,l,r,c : integer);
v = position en y de la souris
p = keyboard status (Touche control_left,control_right, alt, alt_gr, shift_left, shift_right)
h = position en x de la souris
l = touche clavier
r = flags
c = boutons de la souris (0=aucun, 1=gauche, 2=droite, 3=gauche et droite)
Nb: le bouton central n'est pas géré.

NOTEZ BIEN! Lorsque les événements du clavier sont pris en compte dans le gestionnaire, **il ne faut pas** utiliser les fonctions d'entrées clavier readl, readln, hfont, hfont8, hpage, inkey,...) *sous peine de plantage de l'ordinateur*.
}

unit init: **procedure**(checkMouse, checkKeyboard: integer);
{ initializes the Mouse driver.
tells which events will be checked:
if checkMouse <>0 then the events of Mouse will be reported to getpress, see below otherwise ignored;
if checkKeyboard <>0 then the events of Keyboard will be reported to getpress, otherwise ignored
Attention please! While the events of the keyboard are taken under control by *init* or *getmovement*
do not use the functions or procedures: read, readln, hfont, hfont8, hpage, inkey that read keys
YOU RISK TO HANG YOUR SYSTEM!
}
end init

unit getmovement: **procedure**(checkMouse, checkKeyboard: integer);
tells which events will be checked:
if checkMouse <>0 then the events of Mouse will be reported to getpress, see below otherwise ignored;
if checkKeyboard <>0 then the events of Keyboard will be reported to getpress, otherwise ignored
Attention please! While the events of the keyboard are taken under control by *init* or *getmovement*
do not use the functions or procedures: read, readln, hfont, hfont8, hpage, inkey that read keys
YOU RISK TO HANG YOUR SYSTEM!
end getmovement;

unit getpress: **function**(v,p,h,l,r,c : integer): Boolean;
{ *v* = y coordinate of the cursor,
h = x coordinate of the cursor,
p = keybord status control_left,control_right, alt, alt_gr, shift_left, shift_right
l = code of key pressed
r = flags
c = buttons pressed (0=aucun, 1=gauche, 2=droite, 3=gauche et droite)
Nb: the middle button is not taken into account.
end getpress

```
  unit showcursor: procedure;
        {the cursor becomes visible and follows the movements of the mouse}
  end showcursor;

  unit hidecursor: procedure;
        {the cursor becomes invisible}
  end hidecursor;

end MOUSE;
```

Enclosed you find a sample program

```
Program SystemeGraph;
     (* by Frederic Pataud, October 1994 *)
Begin
Pref iiuwgraph block     (* inherit the graphic functions *)
 Begin
 Pref mouse block            (* inherit the mouse functions *)


  (*****************************************************************************)
  (*              P r o g r a m  m e  P r i n c i p a l              *)
  (*****************************************************************************)
  var v,p,h,i : integer,
     l,r,c : integer,
     rep : arrayof char,
     d : boolean,
     xx,yy : arrayof integer,
     status,code,x,y,flags,button : integer;

  Begin

   call gron(0);           (* enter the graphic mode *)
   call init(1,0);          (* initialize the mouse, disregard the keyboard events, check for mouse events *)

   call showcursor;                    (* show cursor *)
   call patern(5,5,635,475,2,0);        (* make a frame around the screen *)
   call outstring(10,10,"x=",2,0);
   call outstring(100,10,"y=",2,0);
   call outstring(10,30,"status = ",2,0);
   call outstring(10,50,"code   = ",2,0);
   call outstring(10,70,"flags  = ",2,0);
   call outstring(10,90,"button = ",2,0);
   call patern(100,210,300,320,3,1);        (* make a rectangle filled in colour 3 *)

   array xx dim (1:6);
   array yy dim (1:6);
   xx(1):=410; yy(1):=10;
   xx(2):=450; yy(2):=30;
   xx(3):=460; yy(3):=50;
   xx(4):=430; yy(4):=80;
   xx(5):=420; yy(5):=40;
   xx(6):=480; yy(6):=30;
   call intens(6,xx,yy,8,1);                    (* show a polygon filled*)
   for i:=1 to 6
    do
     yy(i):=yy(i)+100;
    od;
   call intens(6,xx,yy,15,0);                   (* show another polygon empty *)
```

```
    call cirb(500,300,50,40,100,3500,10,0);  (* draw an empty pie or camembert *)
    call cirb(400,400,40,40,600,4000,11,1);     (* draw a filled pie *)


    i:=hfont(100,350,6,-9999999,9999999,500,9,0,15);      (* read integer from a window *)
    call hpage(100,400,10,unpack("Il fait beau dans ma verte campagne"),9,0);   (* show text *)
    rep:=hfont8(100,430,10,80,unpack("tototutu"),9,0,15);                  (* read text *)

    call getmovement(1,1);          (* take into consideration both key events and mouse events *)

    do
     d:=getpress(v,p,h,l,r,c);                    (* ask about an event *)
     if (d)
     then call outstring(10,400,"Event",2,0);
          call patern(80,25,130,100,0,1);
          call track(40,10,v,0,4);                (* print integer *)
          call track(140,10,p,0,4);
          call track(80,30,h,0,4);
          call track(80,50,l,0,4);
          call track(80,70,r,0,4);
          call track(80,90,c,0,4);
          if((h=164 and l=27) or (c=3))              (* exit if either two buttons were pressed c=3 or Ctrl+Esc key *)
          then exit;
          fi;
     fi;
    od;
    call groff;                                   (* leave the graphic mode and return to the text mode *)
    writeln("i=",i);
    for i:=lower(rep) to upper(rep)
     do
      write(rep(i));
     od;
    writeln;
  End
 End
End.
```