
```

ErSL = 'ErSetLine';
ErSP = 'ErSetPattern';
ErSW = 'ErSetWindow';
ErSV1 = 'Er1SetViewPort';
ErSV2 = 'Er2SetViewPort';
ErPL = 'ErPolyLine';
ErFA = 'ErFillArea';
ErSM = 'ErSetMarker';
ErPM = 'ErSetPolyMarker';
ErSC = 'ErSetColor';
ErGS = 'ErGreateSegment';
ErSV = 'ErSetVisibility';
ErSPR = 'ErSetPriority';
ErSST = 'ErSetSegmentIran';
ErIS1 = 'Er1InsertSegment';
ErIS2 = 'Er2InsertSegment';
ErRS1 = 'Er1RenameSegment';
ErRS2 = 'Er2RenameSegment';
ErDS = 'ErDeleteSegment';
ErPG1 = 'Er1Polygon3';
ErPG2 = 'Er2Polygon3';

```

Mess1 = ' Η εκτέλεση του προγράμματος διακόπτεται. ';

Mess2 = ' Πατήστε Enter για να συνεχίσετε ';

Type

Strin = String[16];

Tex = Text;

{ Πίνακας για τις διαδικασίες Polyline2, Fillarea2, PolyMarker2 }

PolyType = Array[1..MaxNumberOfPointsInPolyline] of Real;

TransAr = Array [1..2,1..3] of Real; { Πίνακας μετασχηματισμών }

TrAr3x3 = Array [1..3,1..3] of Real; { Χρησιμοποιείται σε ενδιάμεσα
στάδια για τους μετασχηματισμούς }

{ Συνδεδεμένη λίστα της οποίας κάθε κόμβος αποθηκεύει μια εντολή του ΓΑΠ μαζί
με τα ορίσματα της }

CompPtr = ^CompType;

CompType = record

ComCode : integer; { Κώδικας Εντολής }

Arguments : Pointer; { Ορίσματα της εντολής }

Next : CompPtr;

end;

Attrib = Record { Εγγραφή για τα Χαρακτηριστικά του τμήματος }

Visibility : Boolean; { Ορατότητα }

Priority : Real; { Προτεραιότητα }

Transfrm : TransAr; { Πίνακας Μετασχηματισμών }

Displayed : Boolean; { Για το Refresh της οθόνης }

end;

SegnType = record { Ορισμός τμήματος }

Name : Integer; { Ονομα }

Attr : Attrib; { Χαρακτηριστικά }

Components, { Δείκτες για τον χειρισμό }

Last : CompPtr; { λίστας δεδομένων }

end;

{-----}

```

Text_Type = Strin;      { Για την εντολή Text }

{ Για τις εντολές με όρισμα έναν ακέραιο:
  SetLine, SetPattern, SetMarker, Set_Color }
One_Integer_Type = Integer;

Two_Real_Type = record   { Για τις εντολές με 2 ορίσματα :
  x, y : Real;   { MoveAbs2, MoveRel2, LineAbs2, LineRel2 }
end;

{ Για τις εντολές με ορίσματα 2 πίνακες και 1 ακέραιο: }
Three_Args_Type = record { PolyLine2, Fillarea, PolyMarker2
  Xar, Yar : PolyType;
  N        : Integer;
end;

Const NormArray : TransAr = (      { Μοναδιαίος Πίνακας }
  ( 1, 0, 0 ),
  ( 0, 1, 0 ));

{-----}
Var
  V_Left,V_Right,V_Top,V_Bottom ,      { Όρια του View Port      }
  W_Left,W_Right,W_Top,W_Bottom : Real; { Όρια του Window        }
  MaxX,MaxY:Integer;                   { Όρια των X και Y σε DC  }
  CurMarker,                           { Τρέχων PolyMarker      }
  Color,                                { Τρέχων Χρώμα           }
  LineStyle,                            { Τρέχων Line Style      }
  GraphDriver,GraphMode      : Integer; { Για την Κάρτα Γραφικών  }
  CPX,CPY                     : Real;   { Τρέχουσα θέση σε NDC   }
  PageCounter                 : Word;    { Περιέχει την Active Page }
  SegmOpen                    : Boolean;  { Δείχνει αν υπάρχει ανοικτό
                                         τμήμα                      }
  { Όνομα του τρέχοντος τμήματος }
  SegmNum                     : 1..MaxSegmentNumber;

  { Πίνακας Περιεχομένων των Τμημάτων }
  Segment                     : Array [1..MaxSegmentNumber] of SegsType;

  ErrFile:Tex;                { Αρχείο Πληροφοριών για τα λάθη }

```

Implementation

```

begin
end.

```

```

(* * * * *
*           Πρόγραμμα : Gap.Pas  Version 1.0           *
*           Υλοποίηση των ρουτίνων του 2D ΓΑΠ.         *
* * * * *
*)

```

Unit Gap;

Interface

Uses

GapDecl, Crt, Graph;

{-----}

```

Procedure WC_to_NDC ( Xw,Yw : Real; var Xn,Yn : Real);
Procedure NDC_to_DC ( Xn,Yn : Real; var Xd,Yd : Integer);
Procedure WC_to_DC ( Xw,Yw : Real; var Xd,Yd : Integer);
Procedure Error ( Message : Strin);
Procedure TerminateProgram;

Procedure CloseDev;
Procedure OpenDev;
Procedure ClearScreen;
Procedure SetViewPort2 ( Left, Right, Bottom, Top : Real);
Procedure SetWindow2 ( Left, Right, Bottom, Top : Real);
Procedure MoveAbs2 ( x, y : Real);
Procedure MoveRel2 ( dx,dy : Real);
Procedure Text ( Stri : String);
Procedure TextAtXY ( X, Y : Real; Stri : Strin);
Procedure LineAbsNdc2 ( x, y : Real );
Procedure LineAbs2 ( x, y : Real );
Procedure LineRel2 ( dx,dy : Real);
Procedure SetLine ( N : Integer );
Procedure Set_Color ( N : Integer);
Procedure SetMarker ( Marker : Integer);
Procedure PolyMarker2 ( Pos_X, Pos_Y : PolyType; Max: Integer);
Procedure SetPattern ( N : Integer);
Procedure Polyline2 ( Pos_X, Pos_Y : PolyType; Max: Integer);
Procedure FillArea2 ( Pos_X, Pos_Y : PolyType; Max: Integer);
Procedure CreateSegment ( Name : integer);
Procedure CloseSegment;
Procedure SetVisibility ( SegName : Integer; Vistate : Boolean);
Procedure SetSegmentTran ( SegmentName: Integer; M : TransAr );
Procedure AccumTran ( A :TransAr; px,py,dx,dy,r,Sx,Sy :real; sw :boolean;
var M : TransAr);
Procedure EvalTran ( px,py,dx,dy,r,Sx,Sy :real; sw :boolean;
var M : TransAr);
Procedure SetPriority ( SegName: Integer; Pr : real);
Procedure InsertSegment ( SegName : integer; A :TransAr);
Procedure RenameSegment ( OldSegmentName, NewSegmentName : integer );
Procedure DeleteSegment ( SegName : Integer);
Procedure Get_Window_Settings ( Var x1, x2, y1, y2 : Real);
Procedure Get_ViewPort_Settings ( Var X1, X2, Y1, Y2 : Real);
Function XDif:real;
Function YDif:real;
Function Get_Color:integer;
Function Get_Line_Style:integer;

```

Implementation

{
ΔΙΑΔΙΚΑΣΙΕΣ ΜΕΤΑΣΧΗΜΑΤΙΣΜΟΥ ΣΥΝΤΕΤΑΓΜΕΝΩΝ

A

```

----- }
Procedure WC_to_NDC( Xw,Yw : Real; { World Coordinates }
                    var Xn,Yn : Real); { Normalized Coordinates }
{ Μετατρέπει τις World Coordinates σε Normalized Device Coordinates }
begin
  Xn := (V_Right - V_Left) * (Xw-W_Left) / (W_Right-W_Left) + V_Left;
  Yn := (V_Top - V_Bottom) * (Yw-W_Bottom) / (W_Top-W_Bottom) + V_bottom;
end;

{-----}

Procedure NDC_to_DC ( Xn,Yn : Real; { Normalized Coordinates }
                    var Xd,Yd : Integer); { Device Coordinates }
{ Μετατρέπει τις normalized device coordinates σε world coordinates }
begin
  Xd := Round( (Xn - V_Left) * MaxX );
  Yd := Round( MaxY * (V_Top - Yn));
end;

{-----}

Procedure WC_to_DC ( Xw,Yw : Real; { World Coordinates }
                    var Xd,Yd : Integer); { Device Coordinates }
{ Μετατρέπει τις world coordinates σε device coordinates }
var
  Xn, Yn : Real;
begin
  WC_to_NDC ( Xw, Yw, Xn, Yn);
  NDC_to_DC ( Xn, Yn, Xd, Yd);
end;

{ Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ   O P E N   /   C L O S E   D E V I C E }
----- }

Procedure CloseDev;
{ Βγάζει το σύστημα από την κατάσταση Γραφικών και το θέτει στη προηγούμενη
  κατάσταση κειμένου (Text Mode) }
begin
  CloseGraph;
  TextMode(LastMode);
end;

Procedure FindError(ErrorMessage:Strin; Var ErrFile:Tex);
{ Η αναδρομική αυτή διαδικασία δέχεται σαν όρισμα τον κωδικό λάθους
  ErrorMessage και αναζητεί στο αρχείο Text ErrFile τις γραμμές που
  αντιστοιχούν στον κωδικό αυτόν. Μόλις βρεθεί η γραμμή με τον κωδι-
  κό ErrorMessage τότε τυπώνονται όλες οι επόμενες γραμμές μέχρι τη
  γραμμή με τον επόμενο κωδικό τυπώνονται στην οθόνη. Όπως αναφέρθηκε
  παραπάνω οι γραμμές αυτές περιέχουν πληροφορίες για την εντολή στην
  οποία βρέθηκε το λάθος. }

var Ch,Ch1:char;
    Line:string[79];
    S:stin;
begin
  If Not EOF(ErrFile) then
    begin
      Read(ErrFile,Ch);
      { Οι γραμμές που περιέχουν κωδικούς αναγνωρίζονται από τον χαρακτή-
        ρα '*' που έχουν στην αρχή τους.}
      If Ch='*' then
        begin
          Readln(ErrFile,S);
          If S = ErrorMessage then { Αν βρέθηκε η ζητούμενη γραμμή τότε
            τυπώνονται οι επόμενες γραμμές}
            begin

```

```

    Ch1:='I';
    While Ch1<>'*' do
        begin
            Readln(ErrFile,Line);
            Ch1:=Line[1];
            If Ch1 <> '*' then Writeln ( Line );
        end;
    end {if}
else { Αλλιώς γίνεται αναζήτηση στην επόμενη γραμμή }
begin { με αναδρομική κλήση της διαδικασίας. }
    Readln(ErrFile);
    FindError(ErrorMessage,ErrFile);
end
end
else
begin
    Readln(ErrFile);
    FindError(ErrorMessage,ErrFile)
end
end;
end;
end;

```

Procedure Error (Message : Strin);
 { Στεματίζει την εκτέλεση του προγράμματος αφού θέσει το σύστημα εκτός
 κατάστασης Γραφικών και καλεί την FindError με κωδικό Mess }

```

begin
    CloseDev;
    Assign(ErrFile,'GAPERR.DAT');
    Reset(ErrFile);
    Clrscr;
    writeln('                R U N   T I M E   E R R O R'); writeln;
    FindError(Message,ErrFile);
    close(ErrFile);
    Writeln('        Η λανθασμένη εντολή είναι η :');
end;

```

Procedure TerminateProgram;

```

Var Ch:Char;
begin
    Writeln; writeln;
    writeln( Mess1 );
    writeln( Mess2 );
    Ch:=ReadKey; Halt(0);
end;

```

Procedure OpenDev;

{ Θέτει το σύστημα σε Κατάσταση Γραφικών. Αν συμβεί κάποιο λάθος τότε το
 πρόγραμμα τερματίζεται και τυπώνεται το ανάλογο μήνυμα }

```

Var
    Flag : Integer;
begin
    DetectGraph(GraphDriver,GraphMode); { Αυτόματα αναγνώριση κάρτας γραφικών }
    InitGraph(GraphDriver,GraphMode,'');
    Flag := GraphResult;
    if (flag <> 0) then { Αν συμβεί λάθος κατά τη μετάβαση
                        στην κατάσταση Γραφικών }
        Error( GraphErrorMsg(GraphResult) )
    else
        begin
            { Προκαθορισμένη τιμή του Window είναι η (0,1,0,1) }
            W_Left := 0; W_Right := 1; W_Top :=1; W_Bottom := 0;
            { Προκαθορισμένη τιμή του ViewPort είναι η (0,1,0,1) }

```

```

V_Left := 0; V_Right := 1; V_Top := 1; V_Bottom := 0;
CPX := 0; CPY := 0; { Προκαθορισμένη Τρέχουσα Θέση : (0,0,0.0) }
MaxX:=GetMaxX; MaxY:=GetMaxY; { R E S O L U T I O N }
SegmOpn := Off; { Αρχικά κανένα τμήμα δεν είναι ανοικτό }
end;
end;

```

```

Procedure ClearScreen;
{ Καθαρίζει το τρέχον View Port }
begin
  ClearViewPort;
end;

```

```

{-----}

```

```

Procedure PutComInSegn ( Opcode : Integer; var Data : Pointer );
{ Εισάγει μια εντολή με Κώδικα OpCode και Ορίσματα Data στο τρέχον τμήμα.
  Η εντολή εισάγεται στο τέλος της λίστας του τρέχοντος τμήματος }
var temp,prt : CompPtr;
    i : byte;
begin
  new (prt);
  prt^.ComCode := Opcode; { Δημιουργία κόμβου }
  prt^.Arguments := Data;
  prt^.next := nil;
  { Ίσυνθεση με τη λίστα }
  Segment [ SegmNum ]. Last^.next := prt ;
  Segment [ SegmNum ]. Last := Segment [ SegmNum ]. Last^.next;
  Segment [ SegmNum ]. Last^.next := nil;
end;

```

```

{ Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ Κ Α Θ Ο Ρ Ι Σ Μ Ο Υ W I N D O W / V I E W P O R T }
{-----}

```

```

Procedure SetViewPort2 (Left,Right,Bottom,Top : Real);
{ Ορίζει το τρέχον View Port. }
Var
  D_Left, D_Right, D_Top, D_Bottom : Real;

```

```

Function IsNDC( Number : Real) : Boolean;
{ Επιστρέφει TRUE αν 0<= Number <= 1 }
begin
  if ( 0 <= Number ) and ( Number <= 1 ) then IsNDC := TRUE
  else IsNDC :=FALSE;
end;


```

```

begin
if not ( IsNDC(Left) and IsNDC(Right) and
        IsNDC(Top) and IsNDC(Bottom)) then
begin
  Error(ErSV1);
writeln('SetViewPort2(',Left:4:1,',',Right:4:1,',',Bottom:4:1,',',Top:4:1,')');
  TerminateProgram;
end;
if (Left > Right) or (Bottom > Top) then
begin
  Error(ErSV2);
writeln('SetViewPort2(',Left:4:1,',',Right:4:1,',',Bottom:4:1,',',Top:4:1,')');
  TerminateProgram;
end;
D_Left := left * MaxX; D_Right := Right * MaxX;
D_Top := (1 - Top) * MaxY; D_Bottom := (1 - Bottom) * MaxY;
SetViewPort(Round(D_Left),Round(D_Top),Round(D_Right),Round(D_Bottom),ClipOn);
V_Left := Left; V_Right := Right;
V_Top := Top; V_Bottom := Bottom;

```

To Viewport is always passed w/ left/right as clipping.



```

end;

Procedure SetWindow2 (Left,Right,Bottom,Top : Real);
{ Ορίζει το τρέχον window }
begin
  if (Left > Right) or (Bottom > Top) then
    begin
      Error(ErSW);
      writeln('SetWindow2( ',left:4:1,' ',right:4:1,' ',bottom:4:1,' ',top:4:1,' ');
      TerminateProgram;
    end;
  W_Left := Left;      W_Right := Right;
  W_Top  := Top;       W_Bottom := Bottom;
end;

```

```

{ Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ Κ Α Θ Ο Ρ Ι Σ Μ Ο Υ Τ Ρ Ε Χ Ο Υ Σ Α Σ Θ Ε Σ Η Σ }
----- }

```

```

Procedure MoveAbsNdc2 ( x,y : Real );      { Τα x,y είναι σε ND Coordinates }
{ Τμήμα της MoveAbs2 το οποίο εκτελεί την MoveAbs2 σε NDC }
var Xd, Yd : Integer;
begin
  CPX := X;          { Καθορίζεται η νέα τρέχουσα θέση }
  CPY := Y;
  NDC_to_DC ( CPX,CPY, Xd,Yd );
  MoveTo ( Xd, Yd );
end;

```

```

Procedure MoveAbs2 ( x,y : Real );
{ Η νέα τρέχουσα θέση καθορίζεται να είναι το σημείο (X,Y) }
Var
  Xn, Yn : Real;
  Args   : ^Two_Real_Type; { Προσωρινή μεταβλητή που χρησιμοποιείται για την
                             εισαγωγή της εντολής στο τμήμα. }
begin
  WC_to_NDC (X, Y, Xn, Yn);      { Μετατρέπει τα x,y σε DC }
  if SegmOpn then                { Αν υπάρχει τμήμα ανοικτό }
    begin
      New ( Args );
      Args^.X := Xn; Args^.Y := Yn;
      { Εισαγωγή της εντολής στο τμήμα αυτό }
      PutComInSegm (1,Pointer (Args) );
    end;
  MoveAbsNdc2 ( Xn, Yn );      { Εκτέλεση της εντολής }
end;

```

```

{-----}

```

```

Procedure MoveRelNdc2 ( dx,dy : Real );      {Τα x,y είναι σε ND Coordinates }
{ Το τμήμα της MoveRel2 το οποίο εκτελεί την εντολή }
var Xd, Yd : Integer;
    x1, y1 : real;
begin
  WC_to_NDC (0,0,x1,y1);
  dx:= dx-x1; dy := dy-y1;
  CPX := CPX + dx;          { Καθορίζεται η νέα τρέχουσα θέση }
  CPY := CPY + dy;
  NDC_to_DC ( CPX,CPY, Xd,Yd );
  MoveTo ( Xd, Yd );
end;

```

```

Procedure MoveRel2 ( dx,dy : Real );
{ Μετακινεί την τρέχουσα θέση στο σημείο (X,Y) σε World Coordinates }
Var
  Xn, Yn : Real;

```



```

Args : ^Two_Real_Type; { Προσωρινή μεταβλητή που χρησιμοποιείται για την
                        εισαγωγή της εντολής στο τμήμα. }

begin
  WC_to_NDC (dx, dy, Xn, Yn);           { Convert to NDC }
  if SegmOpen then                       { Αν υπάρχει τμήμα ανοικτό }
  begin
    New (Args);
    Args^.X := Xn; Args^.Y := Yn;
    { Εισαγωγή της εντολής στο τμήμα αυτό }
    PutComInSegm ( 2, Pointer (Args));
  end;
  MoveRelNdc2 ( Xn, Yn );               { Εκτέλεση της εντολής }
end;

```

```

{ -----
  DRAW TEXT PROCEDURES
  ----- }

```

```

Procedure Text(Stri : String);
{ Τυπώνει το string Stri στην τρέχουσα θέση.}
var Args : ^Text_Type;
begin
  if SegmOpen then                      { Αν υπάρχει τμήμα ανοικτό }
  begin
    New (Args);
    Args^:= Stri;
    { Εισαγωγή της εντολής στο τμήμα αυτό }
    PutComInSegm ( 12, Pointer (Args));
  end;
  MoveTo(GetX,GetY - TextHeight(Stri));
  OutText(Stri);
end;

```

```

{-----}

```

```

Procedure TextAtXY( X,Y      : Real;
                   Stri : Strin);
{ Μετακινεί την τρέχουσα θέση στο σημείο (X,Y) και εμφανίζει το text. }
begin
  MoveAbs2(X,Y);
  Text(Stri);
end;

```

```

{ -----
  Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ Σ Χ Ε Δ Ι Α Σ Μ Ο Υ Γ Ρ Α Μ Μ Ω Ν
  ----- }

```

```

Procedure LineAbsNdc2 ( x,y : Real );           { x,y are on NDC coords }
{ Το τμήμα της LineAbs2 το οποίο εκτελεί την εντολή }
var Xd, Yd : Integer;
begin
  CPX := X;           { Καθορισμός της νέας Τρέχουσας θέσης }
  CPY := Y;
  NDC_to_DC ( x,y, Xd,Yd );
  LineTo (Xd, Yd);
end;

```

```

Procedure LineAbs2 ( x,y : Real );
{ Σχεδιάζει μια γραμμή από την τρέχουσα θέση στο σημείο
  ( x , y ) σε απόλυτες συντεταγμένες. }
Var
  Xn, Yn : Real;
  Args : ^Two_Real_Type; { Προσωρινή μεταβλητή που χρησιμοποιείται για την
                        εισαγωγή της εντολής στο τμήμα. }

```

```

begin
  WC to NDC (X, Y, Xn, Yn);           { Μετατρέπει τα x,y σε DC }

```

```

if Seg#0pn then { Αν υπάρχει τμήμα ανοικτό }
begin
  New(Args);
  Args^.X := Xn; Args^.Y := Yn;
  { Εισαγωγή της εντολής στο τμήμα αυτό }
  PutComInSeg# ( 3, Pointer (Args));
end;

LineAbsNdc2 { Xn, Yn }; { Εκτέλεση της εντολής }
end;

{-----}

Procedure LineRelNdc2 { dx,dy : Real }; { Τα x,y είναι σε NDC }
{ Το μέρος της linerel2 το οποίο εκτελεί την εντολή }
var Xd, Yd : Integer;
    x1, y1 : Real;
begin
  WC_to_NDC (0,0,x1,y1); { Διορθώσεις }
  dx:= dx-x1; dy := dy-y1;
  CPX := CPX + dx; { Καθορισμός τρέχουσας }
  CPY := CPY + dy;

  NDC_to_DC { CPX, CPY, Xd,Yd };
  LineTo (Xd, Yd);
end;

Procedure LineRel2 { dx,dy : Real };
{ Σχεδιάζει μια γραμμή από την τρέχουσα θέση στο σημείο
  (x+dx,y+dy) σε απόλυτες συντεταγμένες. }
Var
  Xn, Yn : Real;
  Args : ^Two_Real_Type; { Προσωρινή μεταβλητή που χρησιμοποιείται για την
                           εισαγωγή της εντολής στο τμήμα. }
begin
  WC_to_NDC (dx, dy, Xn, Yn); { Μετατροπή σε NDC }
  if Seg#0pn then { Αν υπάρχει τμήμα ανοικτό }
  begin
    New (Args);
    Args^.X := Xn; Args^.Y := Yn;
    { Εισαγωγή της εντολής στο τμήμα αυτό }
    PutComInSeg# ( 4, Pointer (Args));
  end;
  LineRelNdc2 { Xn, Yn }; { Εκτέλεση της εντολής }
end;

{-----}

Procedure SetLine ( n : Integer );
{ Καθορίζει το τρέχων Line Style.}
var Style : Integer;
    Args : ^One_Integer_Type;
begin
  if Seg#0pn then { Αν υπάρχει τμήμα ανοικτό }
  begin
    new (Args);
    Args^:= n;
    { Εισαγωγή της εντολής στο τμήμα αυτό }
    PutComInSeg#( 8, Pointer (Args));
  end;
  if N in [1..4] then
    if N in [1..3] then
      begin
        case N of
          1 : Style := SolidLn;
          2 : Style := DashedLn;

```

```

        3 : Style := DottedLn;
    end;
    SetLineStyle ( Style, 7, Norawidth )
end
else SetLineStyle ( UserBitLn, $F1F8, Norawidth )
else
begin
    Error ( ErSL);
    Writeln('SetLine(',N,',')');
    TerminateProgram
end;
end;
end;

```

```

Procedure Set_Color(N:Integer);
var Args: ^One_Integer_Type;
begin
    if SegnOpn then                { Αν υπάρχει τμήμα ανοικτό }
    begin
        new (Args);
        Args^:= n;
        { Εισαγωγή της εντολής στο τμήμα αυτό }
        PutComInSegm( Sett_Color, Pointer (Args));
    end;
    if N in [0..7] then
    begin
        case N of
            BLACK_COLOR   : Color := 0;
            WHITE_COLOR    : Color := 15;
            RED_COLOR      : Color := 4;
            GREEN_COLOR    : Color := 2;
            BLUE_COLOR     : Color := 1;
            CYAN_COLOR     : Color := 3;
            MAGENTA_COLOR  : Color := 5;
            YELLOW_COLOR   : Color := 14;
        end;
        SetColor ( Color )
    end
    else
    begin
        Error ( ErSC);
        Writeln('SetColor(',N,',')');
        TerminateProgram
    end;
end;
end;

```

```

{ Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ Κ Α Θ Ο Ρ Ι Σ Μ Ο Υ   M A R K E R   ( S E T / P U T )
----- }

```

```

Procedure SetMarker(Marker : Integer);
{ Καθορίζει τον τρέχον Marker }
var Args : ^One_Integer_Type;
begin
    if SegnOpn then                { Αν υπάρχει τμήμα ανοικτό }
    begin
        new (Args);
        Args^:= Marker;
        { Εισαγωγή της εντολής στο τμήμα αυτό }
        PutComInSegm(10,Pointer (Args));
    end;
    if Marker in [1..5]
    then CurMarker := Marker
    else
    begin
        Error(ErSM);
        writeln('SetMarker(',Marker,',')');
        TerminateProgram
    end;
end;

```

```

end
end;

{-----}

Procedure PolyMarkerNDC2( Xar,Yar : PolyType;
                        Points : Integer);
{Το τμήμα της PolyMarker2 το οποίο εκτελεί την εντολή.Οι Xar, Yar είναι σε NDC}
Var
  Counter : Integer;
begin
  for Counter := 1 to Points
  do begin
    MoveAbsNDC2 (Xar[Counter],Yar[Counter]);
    MoveTo(GetX-3,GetY);
    Case CurMarker of
      1 : OutText('.');
      2 : OutText('+');
      3 : OutText('#');
      4 : OutText('O');
      5 : OutText('X');
    end { case }
  end { for }
end;

{-----}

Procedure PolyMarker2 ( pos_X, pos_Y : PolyType; Max: Integer);
{ Βάζει τον τρέχων polymarker σε κάθε σημείο των πινάκων pos_X,pos_Y }
var i : integer;
    Args : ^Three_Args_Type;
    tempX, tempY : real;
begin
  If ( Max >= 3 ) and (Max <= MaxNumberOfPointsInPolyline ) then
  begin
    if SegnOpn then new (Args);
    for i := 1 to Max do
    begin
      Wc_to_NDC (pos_X[i], pos_Y[i], tempX, tempY);
      pos_X[i] := tempX;
      pos_Y[i] := tempY;
      if SegnOpn then { Αν υπάρχει τμήμα ανοικτό }
      begin
        Args^.Xar[i] := pos_X[i];
        Args^.Yar[i] := pos_Y[i];
      end;
    end;
    if SegnOpn then
    begin
      Args^.n := Max;
      { Εισαγωγή της εντολής στο τμήμα αυτό }
      PutComInSegn ( 7, Pointer (Args));
    end;
    polyMarkerNDC2 (pos_X, pos_Y, Max);
  end
else
  begin
    Error(ErPM);
    Writeln('PolyMarker(,Xar,Yar,',Max,')');
    TerminateProgram
  end
end;
end;

```

```

{ Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ Γ Ι Α Τ Α Π Ο Λ Υ Γ Ω Ν Α ( D R A W / F I L L )
----- }
Procedure SetPattern ( n : Integer);

```

```

{ Καθορίζει το τρέχων fill pattern. }
var Args : ^One_Integer_Type;
begin
  if SegnOpn then { Αν υπάρχει τμήμα ανοικτό }
  begin
    new (Args);
    Args^:= n;
    { Εισαγωγή της εντολής στο τμήμα αυτό }
    PutComInSegn ( 9, Pointer (Args));
  end;
  if n in [0..7] then
    SetFillPattern( patterns[n], black)
  else
    begin
      Error (ErSP);
      Writeln('SetPattern(',N,')');
      TerminateProgram
    end
  end;
end;

{-----}

Procedure PolylineNDC2 ( pos_X, pos_Y : PolyType; Max: Integer);
{ Το τμήμα της polyline2 το οποίο εκτελεί την εντολή.
  Τα ορίσματα Pos_X, pos_Y είναι σε NDC }
var
  temp : array [1..MaxNumberOfPointsInPolyline] of PointType;
  i, tempx, tempy : Integer;
begin
  for i := 1 to Max do
    begin
      NDC_to_DC ( Pos_X[i], pos_Y[i], tempx, tempy);
      temp[i].x:= tempx;
      temp[i].y:= tempy; { Μετατροπή στα δεδομένα της Pascal }
    end;
  DrawPoly ( Max, temp );
end;

{-----}

Procedure polyline2 ( pos_X, pos_Y : PolyType; Max: Integer);
var i : integer;
    Args : ^Three_Args_Type;
    tempX, tempY : real;
begin
  If ( Max >= 3 ) and (Max <= MaxNumberOfPointsInPolyline ) then
    begin
      if SegnOpn then new (Args);
      for i := 1 to Max do
        begin
          Wc_to_NDC (pos_X[i], pos_Y[i], tempX, tempY);
          pos_X[i] := tempX;      pos_Y[i] := tempY;
          if SegnOpn then { Αν υπάρχει τμήμα ανοικτό }
            begin
              Args^.Xar[i] := pos_X[i];
              Args^.Yar[i] := pos_Y[i];
            end;
        end;
      if SegnOpn then
        begin
          Args^.n := Max;
          { Εισαγωγή της εντολής στο τμήμα αυτό }
          PutComInSegn ( 5, Pointer (Args));
        end;
      polylineNDC2 (pos_X, pos_Y, max)
    end
  else

```

```

begin
  Error(ErPL);
  Writeln('PolyLine2(Xar,Yar,'Max,')');
  TerminateProgram
end
end;

```

```

{-----}

```

```

Procedure FillAreaNDC2 ( pos_X, pos_Y : PolyType; Max: Integer);
{ Γεμίζει την περιοχή που ορίζεται από τα σημεία των πινάκων pos_X και pos_Y
  με το τρέχων fill style και pattern }
Var
  temp : array [1..MaxNumberOfPointsInPolyline] of PointType;
  i, tempX, tempY : Integer;
begin
  for i := 1 to Max do
    begin
      NDC_to_DC ( Pos_X[i], pos_Y[i], tempX, tempY);
      temp[i].x:= tempX;
      temp[i].y:= tempY;      { Μετατροπή στα δεδομένα της Pascal }
    end;
  FillPoly ( Max, temp );
end;

```

```

{-----}

```

```

Procedure FillArea2 ( pos_X, pos_Y : PolyType; Max: Integer);
var i : integer;
    Args : ^Three_Args_Type;
    tempX, tempY : Real;
begin
  if ( pos_X[1] = pos_X[Max] ) and ( pos_Y[1] = pos_Y[Max] ) and ( 3 <= Max )
    and ( Max <= MaxNumberOfPointsInPolyline ) then
    begin
      if SegmOpn then new (Args);
      for i := 1 to Max do
        begin
          Wc_to_NDC (pos_X[i], pos_Y[i], tempX, tempY);
          pos_X[i] := tempX;      pos_Y[i] := tempY;
          if SegmOpn then          { Αν υπάρχει τμήμα ανοικτό }
            begin
              Args^.Xar[i] := pos_X[i];
              Args^.Yar[i] := pos_Y[i];
            end;
        end;
      if SegmOpn then
        begin
          Args^.n := Max;
          { Εισαγωγή της εντολής στο τμήμα αυτό }
          PutComInSegm ( ό, Pointer (Args));
        end;
      FillAreaNDC2 ( pos_X, pos_Y, max);
    end
  else
    begin
      Error (ErFA);
      Writeln('FillArea(Xar,Yar,'Max,')');
      TerminateProgram
    end
end;

```

```

{ Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ Δ Ι Α Χ Ε Ι Ρ Ι Σ Η Σ Τ Μ Η Μ Α Τ Ω Ν
----- }

```

```

Procedure DoTransformation( M : TransAr; X, Y : Real; var NewX, NewY : real);
{ Μετασχηματίζει το σημείο (x,y) σύμφωνα με τον πίνακα M και επιστρέφει το
  νέο σημείο (NewX,NewY) }

```

```

begin
  NewX := M[1,1]*x + M[1,2]*y + M[1,3];      { Μετασχηματισμός του X }
  NewY := M[2,1]*x + M[2,2]*y + M[2,3];      { και του Y }
end;
{-----}
Procedure DoPolyTrans( M: TransAr; Xar, Yar : PolyType; N : Integer;
                      Var NewXar, NewYar : PolyType);

var i:integer;
begin
  for i:=1 to N do
    DoTransformation(M,Xar[i],Yar[i],NewXar[i],NewYar[i])
  end;

Procedure ExecuteCommand ( CommandCode : Integer; Transfrn : TransAr;
                          var NewArgs );
{ Εκτελεί μια εντολή του ΓΑΠ η οποία βρίσκεται σε κάποιο τμήμα }
var newX, newY : Real;
    NewXar, NewYar : PolyType;
begin
  Case CommandCode of
    Move_Abs2 :with Two_Real_Type (NewArgs) do
      begin
        DoTransformation(Transfrn, X, Y, NewX, NewY);
        MoveAbsNDC2 ( NewX, NewY );
      end;
    Move_Rel2 : with Two_Real_Type (NewArgs) do
      begin
        DoTransformation(Transfrn, X, Y, NewX, NewY);
        MoveRelNDC2 ( NewX, NewY );
      end;
    Line_Abs2 : with Two_Real_Type (NewArgs) do
      begin
        DoTransformation(Transfrn, X, Y, NewX, NewY);
        LineAbsNDC2 ( NewX, NewY );
      end;
    Line_Rel2 : with Two_Real_Type (NewArgs) do
      begin
        DoTransformation(Transfrn, X, Y, NewX, NewY);
        LineRelNDC2 ( NewX, NewY );
      end;

    Poly_line2 : with Three_Args_Type (NewArgs) do
      begin
        DoPolyTrans(Transfrn, Xar, Yar, N, NewXar, NewYar );
        PolyLineNDC2 ( NewXar, NewYar, n );
      end;
    Fill_area2 : with Three_Args_Type (NewArgs) do
      begin
        DoPolyTrans(Transfrn, Xar, Yar, N, NewXar, NewYar );
        FillAreaNDC2 ( NewXar, NewYar, n );
      end;
    Poly_Marker2 : with Three_Args_Type (NewArgs) do
      begin
        DoPolyTrans(Transfrn, Xar, Yar, N, NewXar, NewYar );
        PolyMarkerNDC2 ( NewXar, NewYar, n );
      end;

    Set_Line : SetLine ( One_Integer_Type (NewArgs) );
    Set_Pattern : SetPattern ( One_Integer_Type (NewArgs) );
    Set_Marker : SetMarker ( One_Integer_Type (NewArgs) );
    Sett_Color : SetColor ( One_Integer_Type (NewArgs) );
    Textt : Text ( Text_Type (NewArgs) );
  end;
end;
{-----}

Procedure DrawSegment ( Name : integer );

```

```

{ Σχεδιάζει το τμήμα με όνομα Name. }

var prt : CompPtr;
begin
  with Segment [Name] do
    begin
      prt := Components^.next;
      while (prt <> NIL) and Attr.Visibility do { Εκτέλεση όλων των εντολών }
        with prt^ do { που περιγράφουν το τμήμα }
          begin
            ExecuteCommand ( ComCode, Attr.Transform, Arguments^ );
            prt := prt^.next;
          end;
        end;
      end;
    end;

{-----}

Procedure CreateSegment ( Name : integer);
{ Δημιουργία νέου τμήματος }
begin
  if Name in [1..MaxSegmentNumber] then
    begin
      SegmOn := On; { Τιθεται σε ανοικτή κατάσταση }
      SegmNum := Name;
    end
  else
    begin
      Error (Er6S);
      Writeln('CreateSegment(',Name,')');
      TerminateProgram
    end
  end;
end;

Procedure CloseSegment;
{ Θέτει το τρέχον τμήμα σε κλειστή κατάσταση }
begin
  SegmOn := Off; { ==> Δεν υπάρχει τμήμα ανοικτό }
end;

{-----}

Procedure UpdateScreen;
{ Σύμφωνα με το manual της Pascal μια ΣΕΛΙΔΑ(page) είναι μια αρκετά μεγάλη πε-
χή μνήμης, η οποία αποθηκεύει τα περιεχόμενα της οθόνης γραφικών.
Οι εκδόσεις της Pascal 4, 5, 5.5 μας δίνουν τη δυνατότητα χειριζόμαστε τις
σελίδες ως εξείς:
Μπορούμε να εμφανίζουμε τα περιεχόμενα μιας σελίδας ενώ ταυτόχρονα σε κάποια
άλλη (αόρατη) σελίδα μπορούμε να σχεδιάζουμε τη μορφή μιας επόμενης οθόνης.
Ακολουθως μπορούμε να εμφανίσουμε τη δεύτερη σελίδα δίνοντας έτσι την εντύ-
πωση ότι οι δύο διαδοχικές οθόνες σχεδιάζονται ταυτόχρονα.
Οι εντολές που προσφέρει η Pascal για τον χειρισμό των σελίδων είναι οι
SetVisualPage και η SetActivePage.Όλες οι έξοδοι γραφικών κατευθύνονται
στη σελίδα που καθορίζεται από τη SetActivePage ενώ στην οθόνη είναι ορατή
η σελίδα που καθορίζεται από την SetVisualPage.
}

var I : Integer;
    Min, Max : real;
begin
  SetVisualPage ( PageCounter mod 2);
  SetActivePage ( 1 - PageCounter mod 2 );
  ClearScreen;

  { Αρχικά δεν έχει σχεδιασθεί κανένα τμήμα }
  For I := 1 to MaxSegmentNumber do

```



```

Segment[1]. Attr. Displayed := false;

Min := 0;
Repeat
  Max := 2;      { Σχεδίαση με βάση τις προτεραιότητες }
  { Το παρακάτω For-Loop έχει σαν αποτέλεσμα να μένει στο Max η μικρότερη
    προτεραιότητα από όλα τα τμήματα που δεν έχουν εμφανιστεί στην οθόνη. }
  For I := 1 to MaxSegmentNumber do
    With Segment [ I ] do
      begin
        if ( Attr.Priority <= Min ) and (not Attr. Displayed) then
          begin
            DrawSegment ( I );
            Attr. Displayed := true;
          end;
        if ( Attr.Priority < Max ) and ( Not Attr.Displayed ) then
          Max := Attr.Priority;
        end;
      Min := Max;
    until (Max > 1);

  SetVisualPage ( 1 - PageCounter Mod 2);
  INC ( PageCounter );
  If ( PageCounter = 2) then PageCounter := 0;
end;
{-----}
Procedure SetVisibility ( SegmName : integer; Vistate : Boolean);
{ Καθορίζει την ορατότητα ενός τμήματος }
var i : integer;
begin
  if SegmOpn then                { Η ορατότητα δε μπορεί να καθοριστεί }
    begin                        { όταν υπάρχει ανοικτό τμήμα }
      Error (ErSV);
      Writeln('SetVisibility(',SegmName,',',Vistate,')');
      TerminateProgram
    end
  else
    begin
      Segment [ SegmName ]. Attr.Visibility := Vistate;
      UpdateScreen;    { Επανακαθορισμός της οθόνης }
    end;
end;
{-----}

Procedure PrintTrans ( A : TransAr);
var i,j : integer;
begin
  for i := 1 to 2 do
    begin
      for j := 1 to 3 do
        write (a[i,j]:4:3, ' ');
      writeln;
    end;
end;

Procedure Print3x3 ( A : TrAr3x3);
var i,j : integer;
begin
  for i := 1 to 3 do
    begin
      for j := 1 to 3 do
        write (a[i,j]:4:3, ' ');
      writeln;
    end;
end;

```

```

{-----}

Procedure SetSegmentTran ( SegmentName: Integer; M : TransAr );
{ Καθορίζει τον πίνακα μετασχηματισμού ενός τμήματος M σε WC }
begin
  Segment [SegmentName] .Attr .Transfrm := M;
end;

{-----}

Procedure Identity( var Dest : TrAr3x3);
var i,j:integer;
begin
  for i:=1 to 3 do
    for j:=1 to 3 do
      if (i = j) then Dest[i,j]:=1.0
      else Dest[i,j]:=0.0
    end;
  end;

{-----}

Procedure ArrayMul ( var Dest : TrAr3x3; Sour : TrAr3x3);
{ Πολλαπλασιάζει τους πίνακες Dest και Sour και αφήνει το αποτέλεσμα
στον Dest}
var h,j,k : integer;
Temp : TrAr3x3;
begin
  for h := 1 to 3 do
    for j := 1 to 3 do
      begin
        temp[h,j] := 0;
        for k := 1 to 3 do
          temp[h,j] := temp[h,j] + Dest[h,k]*Sour[k,j];
        end;
      Dest := temp;
    end;
  end;

{-----}

Procedure TransferArray (var Dest : TrAr3x3; dx,dy: real);
{ Εκτελεί τον μετασχηματισμό της μεταφοράς στον πίνακα Dest }
var CarryArray : TrAr3x3;
i,j : integer;
begin
  Identity(CarryArray);
  CarryArray[3,1] := dx;
  CarryArray[3,2] := dy;
  ArrayMul (Dest,CarryArray);
end;

{-----}

Procedure RotateArray (var Dest : TrAr3x3; r: real);
{ Εκτελεί τον μετασχηματισμό περιστροφής στον πίνακα Dest }
var RotArray :TrAr3x3;
i,j : integer;
begin
  Identity(RotArray);
  RotArray [ 1 , 1 ] := Cos(r);
  RotArray [ 2 , 1 ] := Sin(r);
  RotArray [ 1 , 2 ] := -RotArray[2,1];
  RotArray [ 2 , 2 ] := RotArray[1,1];
  ArrayMul ( Dest,RotArray);
end;

{-----}

Procedure ScaleArray (var Dest : TrAr3x3; Sx,Sy: real);
{ Εκτελεί τον μετασχηματισμό της κλιμάκωσης στον πίνακα Dest }
var ScaleArray : TrAr3x3;
i,j : integer;
begin

```

```

Identity(ScaleArray);
ScaleArray [ 1 , 1 ] := Sx;
ScaleArray [ 2 , 2 ] := Sy;
ArrayMul (Dest,ScaleArray);
end;
{-----}
Procedure ConvertTo3x3 (Sour : TransAr; var Dest : TrAr3x3);
{ Μετατρέπει έναν 2x3 πίνακα σε έναν 3x3 }
var i,j : integer;
begin
  for i := 1 to 2 do
    for j := 1 to 3 do
      Dest[j,i] := Sour[i,j];
    Dest[1,3] := 0;
    Dest[2,3] := 0;
    Dest[3,3] := 1;
  end;
end;
{-----}
Procedure ConvertToTransArray (Sour : TrAr3x3; var Dest : TransAr);
{ Μετατρέπει έναν 3x3 πίνακα σε έναν 2x3 }
var i,j : integer;
begin
  for i := 1 to 2 do
    for j := 1 to 3 do
      Dest[i,j] := Sour[j,i];
    end;
  end;
end;
{-----}
Procedure AccunTran (A :TransAr; px,py,dx,dy,r,Sx,Sy :real; sw :boolean;
var M : TransAr );
{ A: 0 πίνακας που θα υποστεί τους μετασχηματισμούς.
M: 0 τελικός πίνακας.
px, py : Το σημείο αναφοράς.
dx, dy : Συντελεστές Μεταφοράς.
r : Γωνία περιστροφής (σε Radians )
Sx, Sy : Συντελεστές Κλιμάκωσης.
}
var tempA : TrAr3x3;
var x1, y1 : real;
begin
  if sw then
    begin
      WC_to_NDC ( px,py ,px,py); { Μετατροπή σε NDC }
      WC_to_NDC ( dx,dy ,dx,dy);
      WC_to_NDC ( 0,0 ,x1,y1);
      px := px - x1; py := py - y1;
      dx := dx - x1; dy := dy - y1;
    end;
  ConvertTo3x3 (A, tempA);
  TransferArray (tempA, -px, -py);
  RotateArray (tempA,r);
  ScaleArray (tempA,Sx,Sy);
  TransferArray (tempA,dx,dy);
  TransferArray (tempA, px, py);
  ConvertToTransArray(tempA, M);
end;
{-----}
Procedure EvalTran( px,py,dx,dy,r,Sx,Sy :real; sw :boolean;
var M : TransAr );
{
M : 0 τελικός πίνακας.
px, py : Το σημείο αναφοράς.
dx, dy : Συντελεστές Μεταφοράς.
r : Γωνία περιστροφής (σε Radians )
Sx, Sy : Συντελεστές Κλιμάκωσης.
}
begin

```

```

AccumTran (NormArray, px,py, dx,dy, r, Sx,Sy, sw, M);
end;

```

```

{-----}

```

```

Procedure SetPriority ( SegmName : Integer; Pr : real);
{ Καθορίζει την προτεραιότητα του τμήματος }
begin
  if ( Pr >= 0 ) and ( Pr <= 1 ) then
    Segment [SegmName].Attr.Priority := Pr
  else
    begin
      Error (ErSPR);
      Writeln('SetPriority(',SegmName,',',Pr:4:1,')');
      TerminateProgram
    end
  end;
end;

```

```

{-----}

```

```

Procedure TransformSegm( CommandCode : integer; Transfrm : transAr;
                        Args:pointer; var TransArgs: pointer );
{ Μετασχηματίζει τα δεδομένα του Arg σύμφωνα με τον Transfrm και το
  αποτέλεσμα αφήνεται στο TransArgs }
var i      : integer;
  prt_1    : ^One_Integer_Type; { Δείκτης σε εγγραφή τύπου One_Integer_Type }
  prt_2    : ^Two_Real_Type;    { Δείκτης σε εγγραφή τύπου Two_Real_Type }
  prt_3    : ^Three_Args_Type;  { Δείκτης σε εγγραφή τύπου Three_Args_Type }
  prt_Txt  : ^Text_Type;        { Δείκτης σε εγγραφή τύπου Text_Type }

```

```

begin
  Case CommandCode of
    Move_Abs2,Move_Rel2,
    Line_Abs2,Line_Rel2 :
      begin
        New ( prt_2 );
        with Two_Real_Type (Args^) do
          DoTransformation(Transfrm, X, Y, prt_2^.X, prt_2^.Y);
          { Αντέγραψε τα περιεχόμενα του Data στο prt_2,
            μετασχηματισμένα με τον Transfrm }
          TransArgs := pointer(Prt_2);
        end;

```

```

    Poly_line2 , Fill_Area2,
    Poly_Marker2 : begin
      new ( Prt_3);
      with Three_Args_Type (Args^) do
        begin
          for i := 1 to n do
            DoTransformation(Transfrm, Xar[i], Yar[i],
                          prt_3^.Xar[i], prt_3^.Yar[i] );
          Prt_3^.n := n;
        end;
        TransArgs := pointer(prt_3);
      end;

```

```

    Set_Line , Set_Pattern,
    Set_Marker , Sett_Color:
      begin
        new (prt_1);
        prt_1^ := One_Integer_Type (Args^);
        TransArgs := pointer(prt_1);
      end;

```

```

Textt : begin
  new (prt Txt);

```

```

        prt.Txt^ := Text_Type (Args^);
        TransArgs := pointer(prt.Txt);
    end;

end;

end;

{-----}

Procedure InsertSegment( SegmName : integer; A :TransAr);
var prt      : CompPtr;
    TransData : pointer;
begin
    if not SegmOpn then
        { Για να εκτελεστεί η εντολή πρέπει να υπάρχει τμήμα ανοικτό.}
    begin
        Error(ErIS1);
        writeln('InsertSegment(',SegmName,',',',',A');
        TerminateProgram
    end
    else
        if SegmName > MaxSegmentNumber then
            begin
                Error ( ErIS2 );
                writeln('InsertSegment(',SegmName,',',',',A');
                TerminateProgram
            end
        else
            with Segment[ SegmName ] do
                begin
                    prt := Components^.next;
                    while (prt <> NIL) do
                        with prt^ do
                            begin
                                TransformSegm ( ComCode, A, Arguments , TransData);
                                PutComInSegm ( ComCode,TransData);
                                prt := prt^.next;
                            end;
                        end;
                    end;
                end;
            end;
        end;
    end;

{-----}

Procedure DeleteList( Var Pnt:CompPtr);
Var I:integer;
begin
    If Pnt^.next <> NIL then
        begin
            Pnt:=Pnt^.Next;
            DeleteList(Pnt);
        end
    else
        Dispose(Pnt);
    end;

end;

Procedure DeleteSegment (SegmName : Integer);
begin
    with Segment [SegmName] do
        begin
            Name      := SegmName;
            Attr.Visibility := On;
            Attr.Priority  := 0;
            Attr.Transform := NormArray;
            DeleteList(Components);
            NEW ( Components );
            components^.next := nil;
            last := components;
        end;
    end;
end;

```

```

end;

{-----}

Procedure InitSegs;
  { Θέτει αρχικές τιμές στα πεδία των τμημάτων. }
var i : integer;
begin
  for i := 1 to MaxSegmentNumber do
    with Segment [i] do
      begin
        Name      := i;
        Attr.Visibility := On;
        Attr.Priority  := 0;
        Attr.Transform := NormArray;
        NEW ( Components );
        components^.next := nil;
        last := components;
      end;
    end;
  end;

{-----}

Procedure RenameSegment( OldSegmentName, NewSegmentName : integer );
begin
  Segment [ NewSegmentName ] := Segment[OldSegmentName];
end;

Procedure Get_Window_Settings ( Var X1, X2, Y1, Y2 : Real);
{ Επιστρέφει τα όρια του τρέχοντος Window }
begin
  X1 := W_Left;   X2 := W_Right;
  Y1 := W_Bottom; Y2 := W_Top;
end;

Procedure Get_ViewPort_Settings ( Var X1, X2, Y1, Y2 : Real);
{ Επιστρέφει τα όρια του τρέχοντος View Port }
begin
  X1 := V_Left;   X2 := V_Right;
  Y1 := V_Bottom; Y2 := V_Top;
end;

Function XDif : Real;
{ Επιστρέφει τη διαφορά των ορίων του X - άξονα του τρέχοντος Window }
begin
  XDif := W_Right - W_Left;
end;

Function YDif : Real;
{ Επιστρέφει τη διαφορά των ορίων του Y - άξονα του τρέχοντος Window }
begin
  YDif:=W_Top - W_Bottom;
end;

Function Get_Color : Integer;
{ Επιστρέφει το τρέχον χρώμα }
begin
  Get_Color := Color
end;

Function Get_Line_Style :Integer;
{ Επιστρέφει το τρέχον LineStyle }
begin
  Get_Line_Style := LineStyle;
end;

```

```
begin
  SetMarker(1);
  InitSegs;
  PageCounter := 0;
end.
```

```

(* * * * *
*          Τρισδιάστατη Επέκταση του ΓΑΠ          *
*          Version 1.0                               *
*          Τμήμα Ορισμών και Δηλώσεων              *
* * * * *
*)

```

Unit Gap3decl;

Interface

Uses

GapDecl, Gap, Graph;

```

{-----}

```

Const

```

    Perspective = True;    (* Παράλληλη Προβολή *)
    Parallel    = False;   (* Προοπτική Προβολή *)
    SmallNumber = 0.00001;
    GAP3D_PART  = 30;

```

```

    MaxSegNumber      = 7; { Μέγιστος αριθμός τρισδιάστατων τμημάτων }

```

```

{-----}

```

ΚΩΔΙΚΕΙ ΤΡΙΣΔΙΑΣΤΑΤΩΝ ΕΝΤΟΛΩΝ ΤΟΥ ΓΑΠ

```

{-----}

```

```

    Set_Line3      = 13;
    Set_Color3     = 14;
    Set_Pattern3   = 15;
    Set_Marker3    = 16;
    Move_Abs3      = 17;
    Move_Rel3      = 18;
    Line_Abs3      = 19;
    Line_Rel3      = 20;
    Poly_Line3     = 21;
    Fill_Area3     = 22;
    Poly_Marker3   = 23;

```

Type

(* Πίνακας Τρισδιάστατων Μετασχηματισμών *)

TransAr3 = Packed Array[1..3,1..4] of Real;

Vector = Record (* Διάνυσμα. Τέτοιου τύπου είναι τα VRP,VUV,VPN,κ.α *)

x,y,z:Real;

end;

Comp3Ptr = ^Comp3Type;

Comp3Type = Record

ComCode3 : integer; { Κώδικας Εντολής }

Arguments3 : Pointer; { Ορίσματα της εντολής }

Next : Comp3Ptr;

end;

Attrib3 = Packed Record { Εγγραφή για τα Χαρακτηριστικά του τμήματος }

Visibility : Boolean; { Ορατότητα }

Priority : Real; { Προτεραιότητα }

Transfrm : TransAr3; { Πίνακας Μετασχηματισμών }

end;

Segm3Type = Record { Ορισμός τμήματος }

Name : Integer; { Ονομα }

Attr3 : Attrib3; { Χαρακτηριστικά }

Components, { Δείκτες για τον χειρισμό }

Last : Comp3Ptr; { Λίστας δεδομένων }


```

end;

{-----}

Three_Real_Type = Record   { Για εντολές με ορίσματα 3 πραγματικούς }
    X,Y,Z : Real
end;

Four_Args_Type = Record    { PolyLine3, Fillarea3, PolyMarker3      }
    X_ar, Y_ar, Z_ar :
    PolyType;
    N      : Integer;
end;

{-----}
Var
    CP3x,CP3y,CP3z : Real;      (* Τρέχουσα θέση για 3 διαστάσεις      *)
    VRP,            (* View Reference Point      *)
    COP,            (* Center Of Projection      *)
    DOP,            (* Direction Of Projection   *)
    VUV,            (* View Up Vector            *)
    VPN:Vector;     (* View Plan Normal          *)
    SPX,SPY:Real;    (* Για την παράλληλη προβολή *)
    V_axis,U_axis:Vector; (* Αριστερόστροφοι Άξονες   *)
    ViewMatrix:TransAr3; (* Πίνακας για τον μετασχηματισμό θέας *)
    Unin,Umax,Vmin,Vmax:real; (* Όρια Window στο επίπεδο θέας *)
    Projection:Boolean; (* Για το είδος της προβολής (Προοπτική ή *)
                        (* παράλληλη)                *)
    Segs3Opn : Boolean; (* Δείχνει αν υπάρχει ανοικτό τμήμα *)
    Segs3Num      : 1..MaxSegNumber; (* Όνομα του τρέχοντος τμήματος *)

    (* Πίνακας Περιεχομένων των Τμημάτων *)
    Segment3d      : Array [1..MaxSegNumber] of Segm3Type;

{-----}

Implementation

begin
end.

```


{-----}

Implementation

Procedure PutCom3dInSegn (Opcode : Integer; var Data : Pointer);
(* Εισάγει μια εντολή με Κώδικα Opcode και Ορίσματα Data στο τρέχον τμήμα.
Η εντολή εισάγεται στο τέλος της λίστας του τρέχοντος τμήματος *)

var temp,prt : Comp3Ptr;
i : byte;
begin
new (prt);
prt^.ComCode3 := Opcode; { Δημιουργία κόμβου }
prt^.Arguments3 := Data;
prt^.next := nil;
{ Σύνδεση με τη λίστα }
Segment3d [Segm3Num]. Last^.next := prt;
Segment3d [Segm3Num]. Last := Segment3d [Segm3Num]. Last^.next;
Segment3d [Segm3Num]. Last^.next := nil;
end;

{-----}

Procedure Identit3 (Var M : TransAr3);
(* Ο 3x4 πίνακας M γίνεται ταυτοτικός *)
var i,j:integer;
begin
for i:=1 to 3 do
for j:=1 to 4 do
if (i = j) then M[i,j]:=1.0
else M[i,j]:=0.0
end;
end;

{-----}

Function DotProduct (V1 , V2 : Vector) : Real;
(* Επιστρέφει το εσωτερικό γινόμενο των διανυσμάτων V1 και V2 *)
begin
DotProduct:=V1.x * V2.x + V1.y * V2.y + V1.z * V2.z
end;

{-----}

Procedure CrossProduct (V1, V2 :Vector; Var V3 : Vector);
(* Επιστρέφει το V3 το οποίο είναι το εξωτερικό γινόμενο των V1 , V2 *)
begin
V3.x:=V1.y * V2.z - V2.y * V1.z;
V3.y:=V1.x * V2.z - V2.x * V1.z;
V3.z:=V1.x * V2.y - V1.y * V2.x;
end;

{-----}

Procedure Do3Transformation(M:TransAr3; X,Y,Z:Real; Var TrX,TrY,TrZ:Real);
(* Μετασχηματίζει το (X,Y,Z) σύμφωνα με τον M στο (TrX,TrY,TrZ) *)
begin
TrX := M[1,1] * X + M[1,2] * Y + M[1,3] * Z + M[1,4];
TrY := M[2,1] * X + M[2,2] * Y + M[2,3] * Z + M[2,4];
TrZ := M[3,1] * X + M[3,2] * Y + M[3,3] * Z + M[3,4];
end;

{-----}

Procedure SetVRP (x, y, z : Real);
(* Καθορισμός του View Reference Point *)

```

begin
  VRP.x:=x; VRP.y:=y; VRP.z:=z;
end;

{-----}

Procedure SetVPN ( X, Y, Z : Real);
(* Καθορισμός του ΚΑΝΟΝΙΚΟΠΟΙΗΜΕΝΟΥ View Plane Normal *)
var D:Real;
begin
  D:=SQRT ( SCR ( X ) + SCR ( Y ) + SCR ( Z ) );
  if D <> 0 then
    VPN.x := x / D; VPN.y := Y / D; VPN.z := Z / D;
  end;

{-----}

Procedure SetVUV( X , Y , Z : Real);
(* Καθορισμός του View Up Vector *)
begin
  VUV.x:=X; VUV.y:=Y; VUV.z:=Z;
end;

{-----}

Procedure Make_V_axis;
(* Καθορίζει τον V άξονα που όπως είναι γνωστό ισούται με το εσωτερικό γινόμενο
  των VPN και VUV *)
var Prod:real;
begin
  Prod := DotProduct ( VPN , VUV );
  V_axis.x:=VUV.x - Prod * VPN.x;
  V_axis.y:=VUV.y - Prod * VPN.y;
  V_axis.z:=VUV.z - Prod * VPN.z
end;

{-----}

Procedure Make_U_axis;
(* Καθορίζει τον U άξονα ο οποίος όπως είναι γνωστό ισούται με το εξωτερικό
  γινόμενο των V_axis και VPN. *)
begin
  CrossProduct ( V_axis , VPN , U_axis);
end;

{-----}

Procedure SetProjectionType(P:Boolean);
(* Καθορισμός του είδους της προβολής *)
begin
  Projection := P
end;

{-----}

Procedure SetCOP(u,v,n:Real);
(* Καθορισμός του Center Of Projection *)
begin
  COP.x:=u; COP.y:=v; COP.z:=-n;
end;

{-----}

Procedure SetDOP(u,v,n:Real);
(* Καθορισμός της Direction Of Projection *)
begin

```

```
DOP.x:=u; DOP.y:=v; DOP.z:=n;
SPX:=DOP.x / DOP.z; SPY:=DOP.y / DOP.z
end;
```

```
{-----}
```

```
Procedure SetViewWindow(UUmin,UUmax,VVmin,VVmax:Real);
```

```
(* Καθορισμός του Window στο επίπεδο θέας *)
```

```
begin
```

```
  UUmin:=UUmin;  UUmax:=UUmax;
```

```
  VVmin:=VVmin;  VVmax:=VVmax;
```

```
  SetWindow2(UUmin,UUmax,VVmin,VVmax);
```

```
end;
```

```
{-----}
```

```
Procedure Multipl3(A,B:TransAr3; var C:TransAr3);
```

```
(* C = A * B όπου A , B : 3x4 πίνακες *)
```

```
Var i,j,k : integer;
```

```
  Sum : Real;
```

```
begin
```

```
  for i:=1 to 3 do
```

```
    begin
```

```
      for j:=1 to 3 do
```

```
        begin
```

```
          Sum:=0.0; for k := 1 to 3 do Sum := Sum + A[i,k] * B[k,j];
```

```
          C[i,j]:=Sum
```

```
        end;
```

```
        C[i,4]:=A[i,1] * B[i,4] + A[i,2] * B[i,4] + A[i,3] * B[i,4] + A[i,4]
```

```
      end
```

```
end;
```

```
{-----}
```

```
Procedure MakeViewMatrix;
```

```
(* Δημιουργεί τον πίνακα μετασχηματισμού θέας *)
```

```
begin
```

```
  ViewMatrix[1,1]:=U_axis.x; ViewMatrix[1,2]:=V_axis.x; ViewMatrix[1,3]:=VPN.x;
```

```
  ViewMatrix[2,1]:=U_axis.y; ViewMatrix[2,2]:=V_axis.y; ViewMatrix[2,3]:=VPN.y;
```

```
  ViewMatrix[3,1]:=U_axis.z; ViewMatrix[3,2]:=V_axis.z; ViewMatrix[3,3]:=VPN.z;
```

```
  ViewMatrix[1,4]:= DotProduct ( U_axis , VRP);
```

```
  ViewMatrix[2,4]:= DotProduct ( V_axis , VRP);
```

```
  ViewMatrix[3,4]:= DotProduct ( VPN , VRP)
```

```
end;
```

```
{-----}
```

```
Procedure Make_Parametr; 
```

```
begin
```

```
  Make_V_Axis; (* Δημιουργία του Αριστερόδρογου Αξονα V *)
```

```
  Make_U_axis; (* Δημιουργία του Αριστερόδρογου Αξονα U *)
```

```
  MakeViewMatrix; (* Δημιουργία του Πίνακα Μετασχηματισμού θέας *)
```

```
end;
```

```
{-----}
```

```
Procedure RightToLeftTran( X,Y,Z:Real; Var RX, RY, RZ : Real);
```

```
(* Το σημείο (x,y,z) μετετρέπεται από το δεξιόδρομο στο αριστερόδρομο. *)
```

```
begin
```

```
  Make_Parametr;
```

```
  Do3Transformation ( ViewMatrix , -X, Y, Z, RX, RY, RZ)
```

```
end;
```

```
{-----}
```

```
Procedure Translate3(Var A:TransAr3; dx,dy,dz:real);
```

7
16 4x8cm

16 (10)
12 (10)
19 pointer (10)
12+pointer (16)!

```

(* Στον πίνακα A εφαρμόζεται ο μετασχηματισμός μεταφοράς. *)
begin
  A[1,4] := A[1,4] + dx;
  A[2,4] := A[2,4] + dy;
  A[3,4] := A[3,4] + dz;
end;

```

{-----}

```

Procedure RotateX3(Var A:TransAr3; Th:Real);
(* Στον πίνακα A εφαρμόζεται ο μετασχηματισμός περιστροφής ως προς X. *)
var Rotate:TransAr3;
begin
  Identi3 ( Rotate );
  Rotate[2,2] := Cos(Th);   Rotate[2,3] := Sin(Th);
  Rotate[3,2] := -Rotate[2,3]; Rotate[3,3] := Rotate[2,2];
  Multipl3 ( A , Rotate , A );
end;

```

{-----}

```

Procedure RotateY3(Var A:TransAr3; Th:Real);
(* Στον πίνακα A εφαρμόζεται ο μετασχηματισμός περιστροφής ως προς Y. *)
var Rotate:TransAr3;
begin
  Identi3 ( Rotate );
  Rotate[1,1] := Cos(Th);   Rotate[1,3] := -Sin(Th);
  Rotate[3,1] := -Rotate[1,3]; Rotate[3,3] := Rotate[1,1];
  Multipl3 ( A , Rotate , A );
end;

```

{-----}

```

Procedure RotateZ3(Var A:TransAr3; Th:Real);
(* Στον πίνακα A εφαρμόζεται ο μετασχηματισμός περιστροφής ως προς Z. *)
var Rotate:TransAr3;
begin
  Identi3 ( Rotate );
  Rotate[1,1] := Cos(Th);   Rotate[1,2] := Sin(Th);
  Rotate[2,1] := -Rotate[1,2]; Rotate[2,2] := Rotate[1,1];
  Multipl3 ( A , Rotate , A );
end;

```

{-----}

```

Procedure Scale3(Var A:TransAr3; Sx,Sy,Sz:Real);
(* Στον πίνακα A εφαρμόζεται ο μετασχηματισμός κλιμάκωσης. *)
var Scale : TransAr3;
begin
  Identi3 ( Scale );
  Scale[1,1] := Sx;   Scale[2,2] := Sy;   Scale[3,3] := Sz;
  Multipl3 ( A , Scale , A );
end;

```

{-----}

```

Procedure ParallelTran(X,Y,Z:Real; Var PX,PY:Real);
(* Επιστρέφει την παράλληλη προβολή (PX,PY) του σημείου (x,y,z). *)
begin
  PX := x - z * SPX;
  PY := y - z * SPY;
end;

```

{-----}

```

Procedure PerspectTran( X, Y, Z :Real; Var PX,PY,PZ:Real; Var Flag:boolean);

```

```

(* Καθορισμός της προοπτικής προβολής του σημείου (x,y,z) *)
var D:Real;
begin
  D := COP.z - Z; { Απόσταση στον Z-άξονα από το COP }
  Flag:=false;
  If Abs ( D ) > SmallNumber then
    begin
      Flag := true;
      PX := ( X * COP.z - COP.x * Z ) / D;
      PY := ( Y * COP.z - COP.y * Z ) / D;
      PZ := Z / D
    end;
end;

{-----}

Procedure ProjTrans(X,Y,Z:Real; Var PrX,PrY:Real);
(* Επιστρέφει την προβολή (Παράλληλη ή προοπτική) του σημείου (X,Y,Z) *)
var Pz: real;
    Flags:Boolean;
begin
  If Projection = Perspective then
    PerspectTran (X, Y, Z, PrX, PrY, PZ,Flags)
  else
    ParallelTran ( X, Y, Z, PrX, PrY);
end;
{-----}

Procedure SetLine3 ( N:Integer);
Var Args : ^One_Integer_Type;
begin
  if Seg=30pn then (* Αν υπάρχει τμήμα ανοικτό *)
    begin
      New ( Args );
      Args^ := N;
      (* Εισαγωγή της εντολής στο τμήμα αυτό *)
      PutCom3dInSegm ( Set_Line3 , Pointer ( Args ) );
    end;
  Gap.SetLine(N);
end;
{-----}

Procedure SetColor3 ( N:Integer);
Var Args : ^One_Integer_Type;
begin
  if Seg=30pn then (* Αν υπάρχει τμήμα ανοικτό *)
    begin
      New ( Args );
      Args^ := N;
      (* Εισαγωγή της εντολής στο τμήμα αυτό *)
      PutCom3dInSegm ( Set_Color3 , Pointer ( Args ) );
    end;
  Set_Color(N);
end;
{-----}

Procedure SetPattern3 ( N:Integer);
Var Args : ^One_Integer_Type;
begin
  if Seg=30pn then (* Αν υπάρχει τμήμα ανοικτό *)
    begin
      New ( Args );
      Args^ := N;
      (* Εισαγωγή της εντολής στο τμήμα αυτό *)
      PutCom3dInSegm ( Set_Pattern3 , Pointer ( Args ) );
    end;
  SetPattern(N);
end;
{-----}

```

```

Procedure SetMarker3 ( N:Integer);
Var Args : ^One_Integer_Type;
begin
  if Segm3Opn then                (* Αν υπάρχει τμήμα ανοικτό *)
  begin
    New ( Args );
    Args^ := N;
    (* Εισαγωγή της εντολής στο τμήμα αυτό *)
    PutCom3dInSegm ( Set_Marker3 , Pointer ( Args ) );
  end;
  SetMarker(N);
end;

```

{-----}

{ Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ Κ Α Θ Ο Ρ Ι Σ Μ Ο Υ Τ Ρ Ε Χ Ο Υ Σ Α Σ Θ Ε Σ Η Σ
----- }

```

Procedure MoveAbs3 ( x,y,z : Real );
(* Η νέα τρέχουσα θέση καθορίζεται να είναι το σημείο (X,Y,Z) *)
Var
  Xn, Yn, Zn,                (* Στο Αριστερόστροφο Σύστημα *)
  ProjX, ProjY : Real;        (* Η Προβολή του (Xn,Yn,Zn) *)
  Args : ^Three_Real_Type;    (* Προσωρινή μεταβλητή που χρησιμοποιείται για την
                                εισαγωγή της εντολής στο τμήμα. *)
begin
  CP3x:=x; CP3y:=y; CP3z:=z;
  (* Μετατροπή των x,y,z σε Αριστερόστροφες Συντεταγμένες *)
  RightToLeftTran (X, Y, Z, Xn, Yn, Zn);
  if Segm3Opn then            (* Αν υπάρχει τμήμα ανοικτό *)
  begin
    New ( Args );
    (* Τα δεδομένα αποθηκεύονται σε αριστερόστροφες συντεταγμένες *)
    Args^.X := Xn; Args^.Y := Yn; Args^.Z := Zn;
    (* Εισαγωγή της εντολής στο τμήμα αυτό *)
    PutCom3dInSegm ( Move_Abs3 , Pointer ( Args ) );
  end;
  (* Προβολή του (Xn,Yn,Zn) σύμφωνα με τις τρέχουσες παραμέτρους. *)
  ProjTrans( Xn, Yn, Zn, ProjX, ProjY);
  (* Εκτέλεση του αντίστοιχης διοδιδαστατης εντολής στο επίπεδο θέας *)
  GAP.MoveAbs2 ( ProjX, ProjY );
end;

```

{-----}

```

Procedure MoveRel3 ( dx,dy,dz : Real );
(* Μετακινεί την τρέχουσα θέση στο σημείο (X,Y,Z) σε World Coordinates *)
Var
  Xn, Yn, Zn, ProjX, ProjY : Real;
  Args : ^Three_Real_Type; { Προσωρινή μεταβλητή που χρησιμοποιείται για την
                              εισαγωγή της εντολής στο τμήμα. }
begin
  CP3x:=CP3x + Dx; CP3y:=CP3y + Dy; CP3z:=CP3z + Dz;
  (* Μετατροπή των x,y,z σε Αριστερόστροφες Συντεταγμένες *)
  RightToLeftTran(dx, dy, dz, Xn, Yn, Zn);
  if Segm3Opn then            (* Αν υπάρχει τμήμα ανοικτό *)
  begin
    New (Args);
    (* Τα δεδομένα αποθηκεύονται σε αριστερόστροφες συντεταγμένες *)
    Args^.X := Xn; Args^.Y := Yn; Args^.Z:=Zn;
    (* Εισαγωγή της εντολής στο τμήμα αυτό *)
    PutCom3dInSegm ( Move_Rel3, Pointer (Args));
  end;
  (* Προβολή του (Xn,Yn,Zn) σύμφωνα με τις τρέχουσες παραμέτρους. *)
  ProjTrans( Xn,Yn,Zn,ProjX, ProjY);
  (* Εκτέλεση του αντίστοιχης διοδιδαστατης εντολής στο επίπεδο θέας *)

```



```
GAP.MoveRel2 ( ProjX, ProjY );
end;
```

```
{ Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ Σ Χ Ε Δ Ι Α Σ Μ Ο Υ Γ Ρ Α Μ Μ Ω Ν
----- }
```

```
Procedure LineAbs3 ( x,y,z : Real );
(* Σχεδιάζει μια γραμμή από την τρέχουσα θέση στο σημείο
   ( x , y , z ) σε απόλυτες συντεταγμένες. *)
Var
  Xn, Yn, Zn, ProjX, ProjY : Real;
  Args : ^Three_Real_Type; { Προσωρινή μεταβλητή που χρησιμοποιείται για την
                              εισαγωγή της εντολής στο τμήμα. }
```

```
begin
  (* Μετατροπή των x,y,z σε Αριστερόστροφες Συντεταγμένες *)
  RightToLeftTran (X, Y, Z, Xn, Yn, Zn);
  if Segm3Open then { Αν υπάρχει τμήμα ανοικτό }
  begin
    New(Args);
    Args^.X := Xn; Args^.Y := Yn; Args^.Z:=Zn;
    { Εισαγωγή της εντολής στο τμήμα αυτό }
    PutCom3dInSegm ( Line_Abs3, Pointer (Args));
  end;
  (* Προβολή του (Xn,Yn,Zn) σύμφωνα με τις τρέχουσες παραμέτρους. *)
  ProjTrans( Xn, Yn, Zn, ProjX, ProjY);
  (* Εκτέλεση του αντίστοιχης διοδιδάτατης εντολής στο επίπεδο θέας *)
  GAP.LineAbs2 ( ProjX, ProjY );
end;
```

```
{-----}
```

```
Procedure LineRel3 ( dx,dy,dz : Real );
(* Σχεδιάζει μια γραμμή από την τρέχουσα θέση στο σημείο
   ( x + dx , y + dy , z + dz ) σε απόλυτες συντεταγμένες. *)
Var
  Xn, Yn, Zn, ProjX, ProjY : Real;
  Args : ^Three_Real_Type; { Προσωρινή μεταβλητή που χρησιμοποιείται για την
                              εισαγωγή της εντολής στο τμήμα. }
```

```
begin
  (* Μετατροπή των x,y,z σε Αριστερόστροφες Συντεταγμένες *)
  RightToLeftTran (dx, dy, dz, Xn, Yn, Zn);
  if Segm3Open then { Αν υπάρχει τμήμα ανοικτό }
  begin
    New (Args);
    Args^.X := Xn; Args^.Y := Yn; Args^.Z:=Zn;
    { Εισαγωγή της εντολής στο τμήμα αυτό }
    PutCom3dInSegm ( Line_Rel3, Pointer (Args));
  end;
  (* Προβολή του (Xn,Yn,Zn) σύμφωνα με τις τρέχουσες παραμέτρους. *)
  ProjTrans(Xn, Yn, Zn, ProjX, ProjY);
  (* Εκτέλεση του αντίστοιχης διοδιδάτατης εντολής στο επίπεδο θέας *)
  GAP.LineRel2 ( ProjX, ProjY );
end;
```

```
{ Δ Ι Α Δ Ι Κ Α Σ Ι Ε Σ Γ Ι Α Τ Α Π Ο Λ Υ Γ Ω Ν Α ( D R A W / F I L L )
----- }
```

```
Procedure PolyLine3 ( Xar, Yar, Zar : PolyType; N: Integer);
var i : integer;
  Args : ^Four_Args_Type;
  TempX, TempY, TempZ : real;
  ProjXar, ProjYar: PolyType; { Πολύγωνα Προβολής των Xar,Yar,Zar *}
begin
  If ( N >= 3 ) and ( N <= MaxNumberOfPointsInPolyline ) then
  begin
```

```

if Segm3Opn then new (Args);
for i := 1 to N do
begin
  RightToLeftTran (Xar[i], Yar[i], Zar[i], TempX, TempY, TempZ);
  Xar[i] := TempX; Yar[i] := TempY; Zar[i] := TempZ;
  if Segm3Opn then { Αν υπάρχει τμήμα ανοικτό }
  begin
    Args^.X_ar[i] := Xar[i];
    Args^.Y_ar[i] := Yar[i];
    Args^.Z_ar[i] := Zar[i];
  end;
end;
if Segm3Opn then
begin
  Args^.N := N;
  { Εισαγωγή της εντολής στο τμήμα αυτό }
  PutCom3dInSegm ( Poly_Line3, Pointer (Args));
end;
(* Προβολή των σημείων του πολυγώνου σύμφωνα με τις τρέχουσες παραμέτρους. *)
For I:=1 to N do
  ProjTrans(Xar[i],Yar[i],Zar[i],ProjXar[i],ProjYar[i]);
(* Εκτέλεση του αντίστοιχης δισδιάστατης εντολής στο επίπεδο θέας *)
GAP.PolyLine2 (ProjXar, ProjYar, N)
end
else
begin
  Error(ErPL);
  Writeln('PolyLine3(Xar,Yar,Zar,',N,',')');
  TerminateProgram
end
end;

```

```

Function Check( X, Y, Z, X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3 : Real) : Boolean;
(* Ελέγχει αν το σημείο (X,Y,Z) ανήκει στο επίπεδο που ορίζεται από τα σημεία
(X1,Y1,Z1), (X2,Y2,Z2), (X3,Y3,Z3). Αν ανήκει επιστρέφει True. *)
Var Temporary:Real;
begin
  Temporary:= X * ( ( Y2 - Y1 ) * ( Z3 - Z1 ) - ( Y3 - Y1 ) * ( Z2 - Z1 ) )
    - Y * ( ( X2 - X1 ) * ( Z3 - Z1 ) - ( X3 - X1 ) * ( Z2 - Z1 ) )
    + Z * ( ( X2 - X1 ) * ( Y3 - Y1 ) - ( X3 - X1 ) * ( Y2 - Y1 ) );
  Check := ( Abs ( Temporary ) <= 0.00001 );
end;

```

$$\begin{array}{c|c} Y_2 - Y_1 & X_3 - X_1 \\ X_2 - X_1 & Z_2 - Z_1 \\ Z_3 - Z_1 & \\ Y_3 - Y_1 & \end{array}$$

```

Procedure Polygon3(Xar,Yar,Zar:PolyType; N:Integer);
Var I:Integer;
    Same:Boolean;
begin
  Same:=True;
  If N >= 4 then
  begin
    I:=4;
    While ( I<=N ) and Same do { Ελεγχος αν όλα τα σημεία βρίσκονται }
    Begin { στο ίδιο επίπεδο. }
      Same := Check ( Xar[I], Yar[I], Zar[I], Xar[1], Yar[1], Zar[1],
        Xar[2], Yar[2], Zar[2], Xar[3], Yar[3], Zar[3]);
      I:=I+1;
    end;
  end;
  If Same then { Ελεγχος αν τα ακραία σημεία ταυτίζονται. }
  If (Xar[1] = Xar[N]) and (Yar[1] = Yar[N]) then
    PolyLine3(Xar,Yar,Zar,N)
  else
  begin
    Error(ErPG1);
    Writeln('Polygon3(Xar,Yar,Zar,',N,',')');
    TerminateProgram
  end;
end;

```

add-poly → αυτή δε ελέγχει αν τα
σημεία είναι στο ίδιο επίπεδο.

```

end
else
begin
  Error(ErP62);
  Writeln('Polygon3(Xar,Yar,Zar,',N,')');
  TerminateProgram
end
end;
{-----}

Procedure FillArea3 ( Xar, Yar, Zar : PolyType; N: Integer);
var i : integer;
    Args : ^Four_Args_Type;
    tempX, tempY, tempZ : Real;
    ProjXar, ProjYar : PolyType;    (* Πολύγωνα Προβολής των Xar,Yar,Zar *)

begin
  if ( Xar[1] = Xar[N] ) and ( Yar[1] = Yar[N] ) and ( Zar[1] = Zar[N] ) and
    ( 3 <= N ) and ( N <= MaxNumberOfPointsInPolyline ) then
    begin
      if Segm3Opn then new (Args);
      for i := 1 to N do
        begin
          RightToLeftTran (Xar[i], Yar[i],Zar[i], tempX, tempY,tempZ);
          Xar[i] := tempX;  Yar[i] := tempY;  Zar[i]:=tempZ;
          if Segm3Opn then      { Αν υπάρχει τμήμα ανοικτό }
            begin
              Args^.X_ar[i] := Xar[i];
              Args^.Y_ar[i] := Yar[i];
              Args^.Z_ar[i] := Zar[i];
            end;
        end;
      if Segm3Opn then
        begin
          Args^.n := N;
          { Εισαγωγή της εντολής στο τμήμα αυτό }
          PutCom3dInSegm ( Fill_Area3, Pointer (Args));
        end;
      (* Προβολή των σημείων του πολυγώνου σύμφωνα με τις τρέχουσες παραμέτρους. *)
      For I:=1 to N do
        ProjTrans(Xar[i],Yar[i],Zar[i],ProjXar[i],ProjYar[i]);
      (* Εκτέλεση του αντίστοιχης δισδιάστατης εντολής στο επίπεδο θέας *)
      GAP.FillArea2 ( ProjXar, ProjYar, N);
    end
  else
    begin
      Error (ErFA);
      Writeln('FillArea3{Xar,Yar,',N,')');
      TerminateProgram
    end
  end;
end;

```

```

Procedure PolyMarker3 ( Xar , Yar , Zar : PolyType; N : Integer);
var i : integer;
    Args : ^Four_Args_Type;
    tempX, tempY, tempZ : Real;
    ProjXar, ProjYar : PolyType;    (* Πολύγωνα Προβολής των Xar,Yar,Zar *)

begin
  if ( 3 <= N ) and ( N <= MaxNumberOfPointsInPolyline ) then
    begin
      if Segm3Opn then new (Args);
      for i := 1 to N do
        begin
          RightToLeftTran (Xar[i], Yar[i],Zar[i], tempX, tempY,tempZ);
          Xar[i] := tempX;  Yar[i] := tempY;  Zar[i]:=tempZ;
          if Segm3Opn then      { Αν υπάρχει τμήμα ανοικτό }

```

```

begin
  Args^.X_ar[i] := Xar[i];
  Args^.Y_ar[i] := Yar[i];
  Args^.Z_ar[i] := Zar[i];
end;
end;
if Segm3Dpn then
begin
  Args^.n := N;
  { Εισαγωγή της εντολής στο τμήμα αυτό }
  PutCom3dInSegm ( Fill_Area3, Pointer (Args));
end;
(* Προβολή των σημείων του πολυγώνου σύμφωνα με τις τρέχουσες παραμέτρους.*)
For I:=1 to N do
  ProjTrans(Xar[i],Yar[i],Zar[i],ProjXar[i],ProjYar[i]);
  (* Εκτέλεση του αντίστοιχης διαδοχικής εντολής στο επίπεδο θέας *)
  GAP.PolyMarker2 ( ProjXar, ProjYar, N);
end
else
begin
  Error (ErPM);
  Writeln('PolyMarker3(Xar,Yar,',N,')');
  TerminateProgram
end
end;
end;

```

```

(* Τρισδιάστατες εκδόσεις των διαδοχικών εντολών *)
{-----}
Procedure MoveAbs2(x,y:real);
begin
  MoveAbs3(x,y,CP3z);
end;
{-----}
Procedure MoveRel2(dx,dy:real);
begin
  MoveRel3(dx,dy,CP3z);
end;
{-----}
Procedure LineAbs2(x,y:Real);
begin
  LineAbs3(x,y,CP3z);
end;
{-----}
Procedure LineRel2(dx,dy:Real);
begin
  LineRel3(dx,dy,CP3z);
end;
{-----}
Procedure PolyLine2(Xar,Yar:PolyType; N:Integer);
var i:integer;
    Zar:PolyType;
begin
  for i:=1 to n do
    Zar[i]:=CP3z;
  PolyLine3(Xar,Yar,Zar,N);
end;
{-----}
Procedure FillArea2(Xar,Yar:PolyType; N:Integer);
var i:integer;
    Zar:PolyType;
begin
  for i:=1 to n do
    Zar[i]:=CP3z;
  FillArea3(Xar,Yar,Zar,N);
end;

```

```
{ ΔΙΑΔΙΚΑΣΙΕΣ ΔΙΑΧΕΙΡΙΣΗΣ ΤΜΗΜΑΤΩΝ
----- }
```

```
Procedure Copy ( SegName:integer);
{ Η διαδικασία αυτή δημιουργεί το διοδιδάστατο αντίγραφο του τμήματος με όνομα
  SegName. Για να μην υπάρχει σύγχυση με τα τμήματα που διαχειρίζεται η
  διοδιδάστατη έκδοση του ΓΑΠ θεωρούμε ότι ο πίνακας περιεχομένων των διοδι-
  δάστατων τμημάτων χωρίζεται σε δύο μέρη, ένα για τα διοδιδάστατα τμήματα του
  ΓΑΠ και ένα για τα διοδιδάστατα αντίγραφα του ΓΑΠ. Ο διαχωρισμός των δύο
  τμημάτων γίνεται με βάση τη σταθερά GAP3D_PART. }
var NewX, NewY, NewZ, ProjX, ProjY : Real;
    NewXar, NewYar, NewZar, ProjXar, ProjYar : PolyType;
    I : Integer;
    Ptr : Comp3Ptr;
begin
  CreateSegment ( GAP3D_PART + SegName ); (* Ενα διοδιδάστατο τμήμα ανοίγει *)
  With Segment3d[ SegName ] do
    begin
      Ptr := Components^.next;
      while (Ptr <> NIL) do
        begin
          with ptr^ do
            Case ConCode3 of
              Move_Abs3 :
                begin
                  With Three_Real_Type(Arguments3^) do
                    (* Τρισδιάστατοι Μετασχηματισμοί *)
                    Do3Transformation(Attr3.Transfrm, X, Y, Z, NewX, NewY, NewZ);
                    (* Προβολή *)
                    ProjTrans( NewX, NewY, NewZ, ProjX, ProjY);
                    (* Εκτέλεση της αντίστοιχης διοδιδάστατης εντολής. *)
                    GAP.MoveAbs2 ( ProjX, ProjY );
                  end;
                Move_Rel3 :
                  begin
                    With Three_Real_Type(Arguments3^) do
                      (* Τρισδιάστατοι Μετασχηματισμοί *)
                      Do3Transformation(Attr3.Transfrm, X, Y, Z, NewX, NewY, NewZ);
                      (* Προβολή *)
                      ProjTrans( NewX, NewY, NewZ, ProjX, ProjY);
                      (* Εκτέλεση του αντίστοιχης διοδιδάστατης εντολής στο επίπεδο θέας *)
                      GAP.MoveRel2 ( ProjX, ProjY );
                    end;
                  Line_Abs3 :
                    begin
                      With Three_Real_Type(Arguments3^) do
                        (* Τρισδιάστατοι Μετασχηματισμοί *)
                        Do3Transformation(Attr3.Transfrm, X, Y, Z, NewX, NewY, NewZ);
                        ProjTrans( NewX, NewY, NewZ, ProjX, ProjY);
                        GAP.LineAbs2 ( ProjX, ProjY );
                      end;
                    Line_Rel3 :
                      begin
                        With Three_Real_Type(Arguments3^) do
                          (* Τρισδιάστατοι Μετασχηματισμοί *)
                          Do3Transformation(Attr3.Transfrm, X, Y, Z, NewX, NewY, NewZ);
                          (* Προβολή *)
                          ProjTrans( NewX, NewY, NewZ, ProjX, ProjY);
                          (* Εκτέλεση του αντίστοιχης διοδιδάστατης εντολής στο επίπεδο θέας *)
                          GAP.LineRel2 ( ProjX, ProjY );
                        end;
                      Poly_line3 :
                        begin
                          with Four_Args_Type (Arguments3^) do
```

```

begin
For i:=1 to N do
begin
  (* Τριβιδόστατοι Μετασχηματισμοί *)
  Do3Transformation(Attr3.Transform, X_ar[i], Y_ar[i], Z_ar[i],
                    NewXar[i], NewYar[i], NewZar[i]);

  (* Προβολή *)
  ProjTrans(NewXar[i], NewYar[i], NewZar[i], ProjXar[i], ProjYar[i]);
end;
GAP.PolyLine2 ( ProjXar, ProjYar, N );
end;
end;
Fill_area3 :
begin
with Four_Args_Type (Arguments3^) do
begin
For i:=1 to N do
begin
  (* Τριβιδόστατοι Μετασχηματισμοί *)
  Do3Transformation(Attr3.Transform, X_ar[i], Y_ar[i], Z_ar[i],
                    NewXar[i], NewYar[i], NewZar[i]);

  (* Προβολή *)
  ProjTrans(NewXar[i], NewYar[i], NewZar[i], ProjXar[i], ProjYar[i]);
end;
  (* Εκτέλεση του αντίστοιχης διβιδόστατης εντολής στο επίπεδο θέας *)
  GAP.FillArea2 ( ProjXar, ProjYar, N );
end;
end;
Poly_Marker3 :
begin
with Four_Args_Type (Arguments3^) do
begin
For i:=1 to N do
begin
  Do3Transformation(Attr3.Transform, X_ar[i], Y_ar[i], Z_ar[i],
                    NewXar[i], NewYar[i], NewZar[i]);

  ProjTrans(NewXar[i], NewYar[i], NewZar[i], ProjXar[i], ProjYar[i]);
end;
  (* Εκτέλεση του αντίστοιχης διβιδόστατης εντολής στο επίπεδο θέας *)
  GAP.PolyMarker2 ( ProjXar, ProjYar, N );
end;
end;
Set_Line3      : Gap.SetLine  ( One_Integer_Type ( Arguments3^ ) );
Set_Pattern3   : Gap.SetPattern( One_Integer_Type ( Arguments3^ ) );
Set_Marker3    : Gap.SetMarker ( One_Integer_Type ( Arguments3^ ) );
Set_Color3     : Gap.Set_Color ( One_Integer_Type ( Arguments3^ ) );
end; { Case }
Ptr:=Ptr^.next;
end; { While }
end;
CloseSegment;  (* Το αντίγραφο δημιουργήθηκε *)
end;

```

{-----}

```

Procedure CreateSegment3 ( Name : integer);
{ Δημιουργία νέου τριβιδόστατου τμήματος }
begin
if Name in [1..MaxSegNumber] then
begin
  Segm3Open := On;          { Τιθεται σε ανοικτή κατάσταση }
  Segm3Num := Name;
end
else
begin
  Error (ErGS);

```

```

    WriteLn('CreateSegment3(',Name,')');
    TerminateProgram
end
end;
{-----}
Procedure CloseSegment3;
(* Θέτει το τρέχων τμήμα σε κλειστή κατάσταση *)
begin
    Segm3Opn := Off;          { ==> Δεν υπάρχει τμήμα ανοικτό }
end;
{-----}
Procedure SetVisibility3 ( SegmName : integer; Vistate : Boolean);
(* Καθορίζει την ορατότητα ενός τμήματος *)
var i,j : integer;
    A : TransAr;
begin
    if Segm3Opn then          { Η ορατότητα δε μπορεί να καθοριστει }
    begin                     { όταν υπάρχει ανοικτό τμήμα }
        Error (ErSV);
        WriteLn('SetVisibility3(',SegmName,',',Vistate,')');
        TerminateProgram
    end
    else
        With Segment3d [ SegmName ].Attr3 do
            begin
                Visibility := Vistate;
                (* Το προηγούμενο αντίγραφο εθάνεται *)
                DeleteSegment( GAP3D_PART + SegmName );
                SetVisibility( GAP3D_PART + SegmName , OFF );
                (* Δημιουργείται νέο αντίγραφο *)
                Copy ( SegmName );
                SetPriority ( GAP3D_PART + SegmName, Priority);
                SetVisibility ( GAP3D_PART + SegmName, Visibility);
            end;
        end;
end;
{-----}
Procedure SetSegmentTran3 ( SegmentName: Integer; M : TransAr3 );
(* Καθορίζει τον πίνακα μετασχηματισμού ενός τμήματος M σε WC *)
begin
    Segment3d [ SegmentName ].Attr3 .Transfrn := M;
end;
{-----}
Procedure SetPriority3 ( SegmName : Integer; Pr : real);
(* Καθορίζει την προτεραιότητα του τμήματος *)
begin
    if ( Pr >= 0 ) and ( Pr <= 1 ) then
        Segment3d [SegmName].Attr3.Priority := Pr
    else
        begin
            Error (ErSPR);
            WriteLn('SetPriority3(',SegmName,',',Pr:4:1,')');
            TerminateProgram
        end
    end;
end;
{-----}
Procedure Transform3Segs ( CommandCode : integer; Transfrn : TransAr3;
    Args:pointer; var TransArgs: pointer );
(* Μετασχηματίζει τα δεδομένα του Arg σύμφωνα με τον Transfrn και το
    αποτέλεσμα αφήνεται στο TransArgs *)

```

```

var i      : integer;
prt_ML    : ^Three_Real_Type; { Δείκτης σε εγγραφή τύπου One_Integer_Type }
prt_PF    : ^Four_Args_Type;  { Δείκτης σε εγγραφή τύπου Two_Real_Type  }

begin
  Case CommandCode of
    Move_Abs3,Move_Rel3,
    Line_Abs3,Line_Rel3 :
      begin
        New ( prt_ML );
        with Three_Real_Type (Args^) do
          Do3Transformation(Transfrm, X, Y, Z,prt_ML^.X, prt_ML^.Y,prt_ML^.Z);
          (* Αντέγραψε τα περιεχόμενα του Data στο prt_ML,
             μετασχηματισμένα με τον Transfrm *)
          TransArgs := pointer(Prt_ML);
        end;

      Poly_Line3 , Fill_Area3,
      Poly_Marker3 : begin
        new ( Prt_PF);
        with Four_Args_Type (Args^) do
          begin
            for I := 1 to n do
              Do3Transformation(Transfrm, X_ar[i], Y_ar[i],Z_ar[i],
                Prt_PF^.X_ar[i], prt_PF^.Y_ar[i],prt_PF^.Z_ar[i] );
              Prt_PF^.n := n;
            end;
            TransArgs := pointer(prt_PF);
          end;
        end;

      end;
    end;
  {-----}

  Procedure InsertSegment3( SegmName : integer; A :TransAr3);
  var prt      : Comp3Ptr;
      TransData : pointer;
  begin
    (* Για να εκτελεστεί η εντολή πρέπει να υπάρχει τμήμα ανοικτό. *)
    if Not Segm3Opn then
      begin
        Error(ErIS1);
        writeln('InsertSegment3(',SegmName,',','A');
        TerminateProgram
      end
    else
      if SegmName > MaxSegmentNumber then
        begin
          Error ( ErIS2 );
          writeln('InsertSegment3(',SegmName,',','A');
          TerminateProgram
        end
      else
        with Segment3d[ SegmName ] do
          begin
            prt := Components^.next;
            while (prt <> NIL) do
              with prt^ do
                begin
                  Transform3Segm ( CosCode3, A, Arguments3 , TransData);
                  PutCom3dInSegm ( CosCode3,TransData);
                  prt := prt^.next;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
end;

```